PROGRAMMING: THE “WHAT”, “WHY” AND “HOW”

Anthony Tung
School of Computing
National U. of Singapore

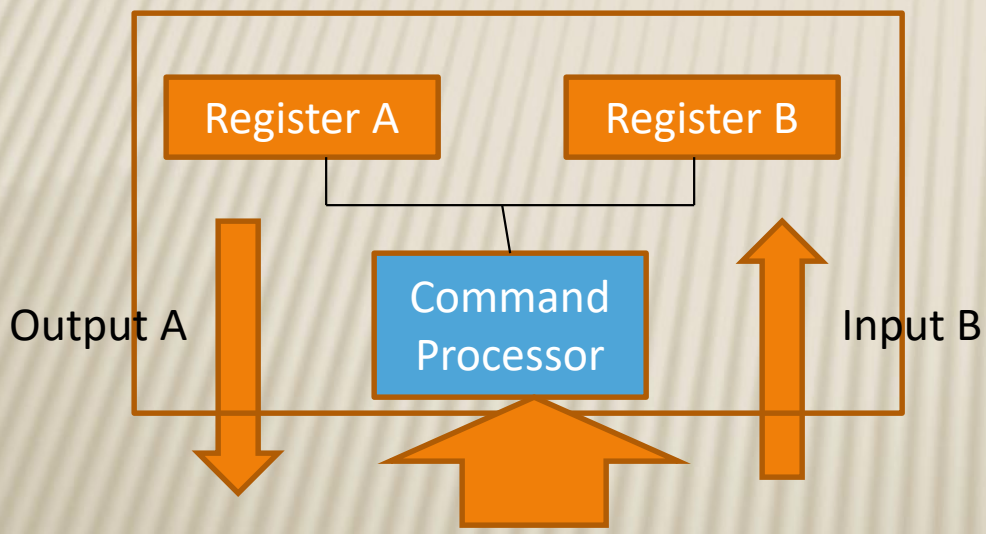
https://t.me/joinchat/H55maBM12a2HFdIU-4kHHg

OBJECTIVES

- Explain what exactly is programming by
  - Designing increasingly more complex machines which can solve increasingly complex problems through increasingly complex programming concepts
  - Through this process, you should be able to appreciate NOT only WHAT programming is but is also able to appreciate WHY certain programming concepts are designed in certain way
- Give a high level road map of the course so that:
  - You don’t get lost in the details later in the course
  - You not only learn a programming language (“C”), but also learn how to learn ANY new programming language

COMPUTING MACHINE A(CALCULATOR): MANUAL INPUT

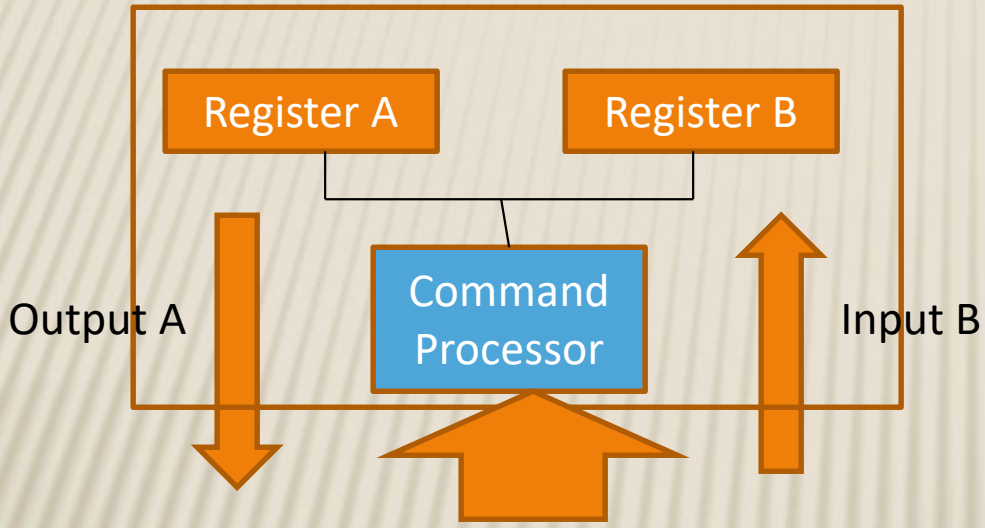
- Consider a simple machine A, with
  - a very simple architecture and
  - a very simple instruction set



Manual input of command one at a time

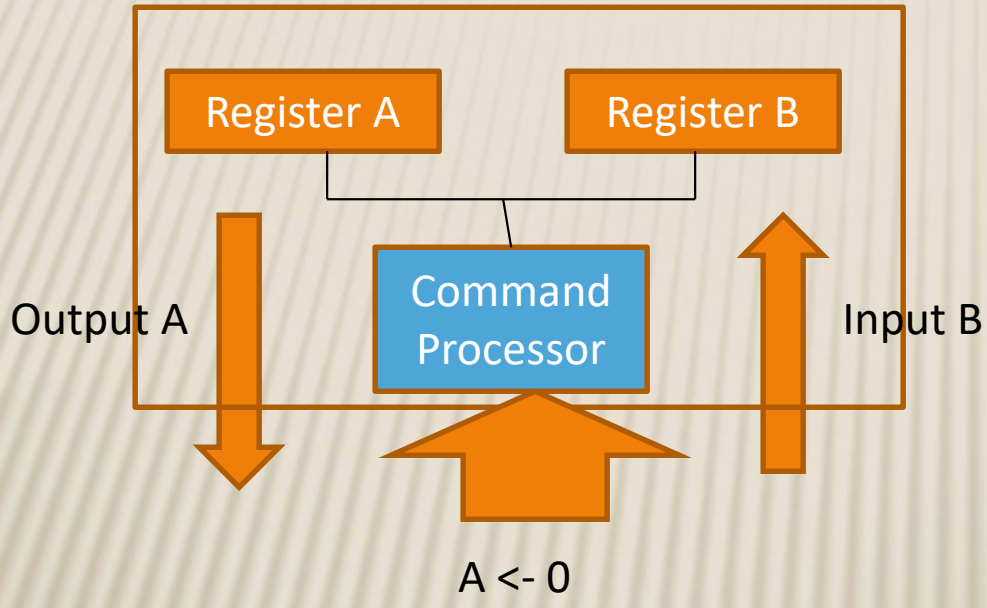
Instruction	Effect
InputB	Read a number into Register B
OutputA	Output the number in Register A
A <- v	Set Register A to value v
B <- v	Set Register B to value v
A <- A + B	Sum the values of Register A and B and put the result into A
A <- A/B	Divide the values of Register A and by the value in Register B and put the result into A

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



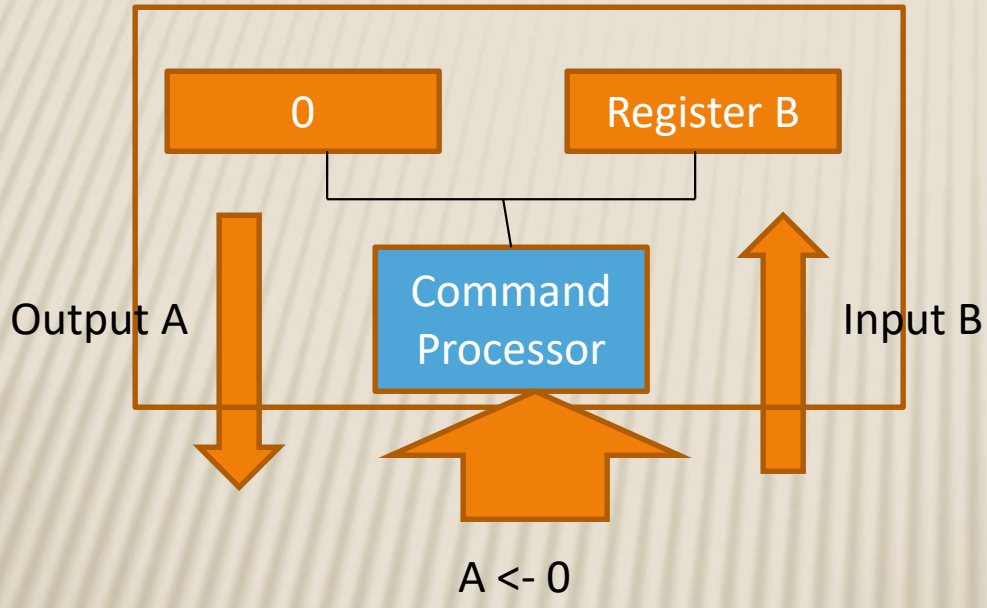
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



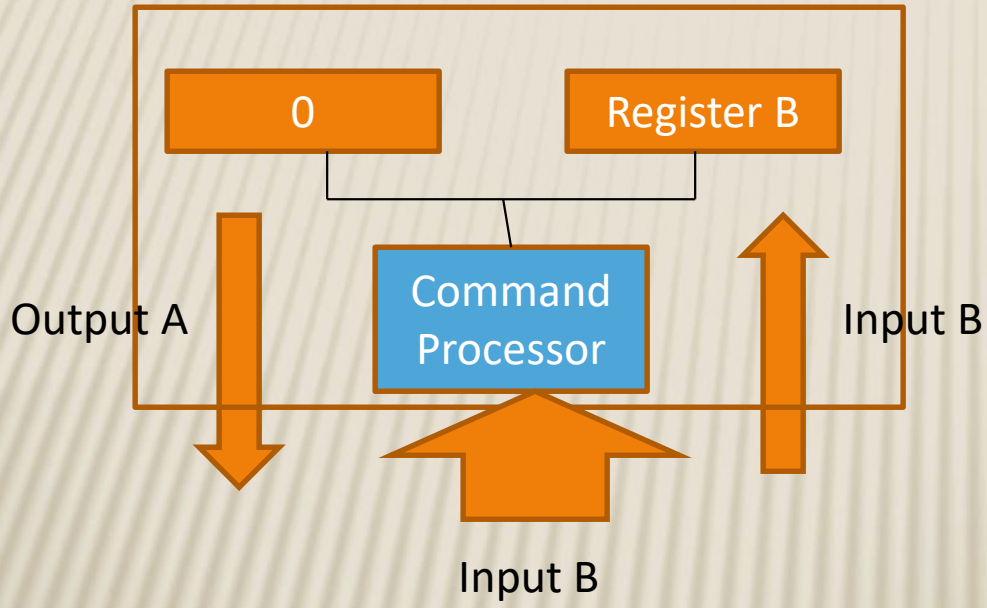
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



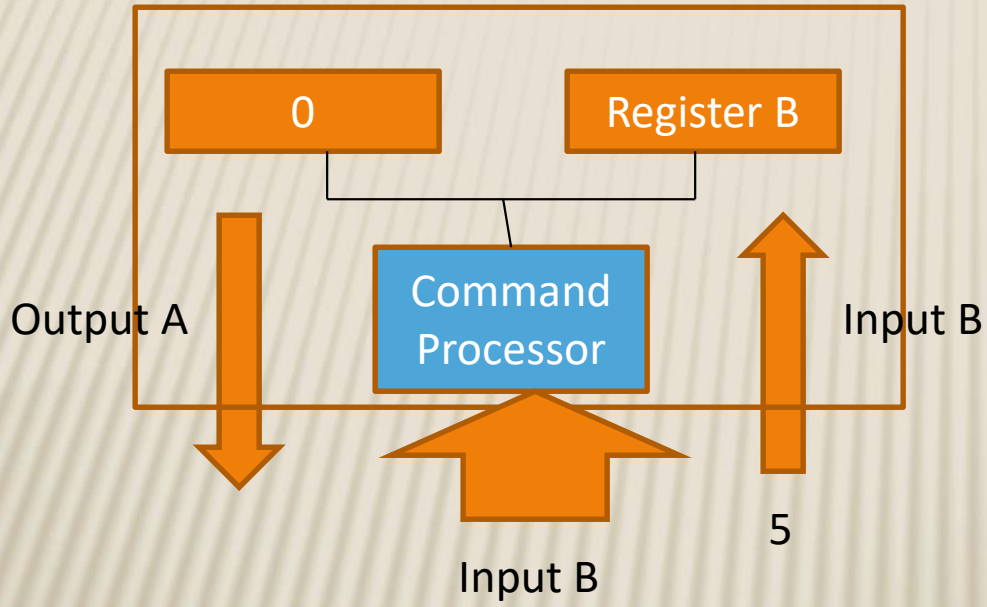
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



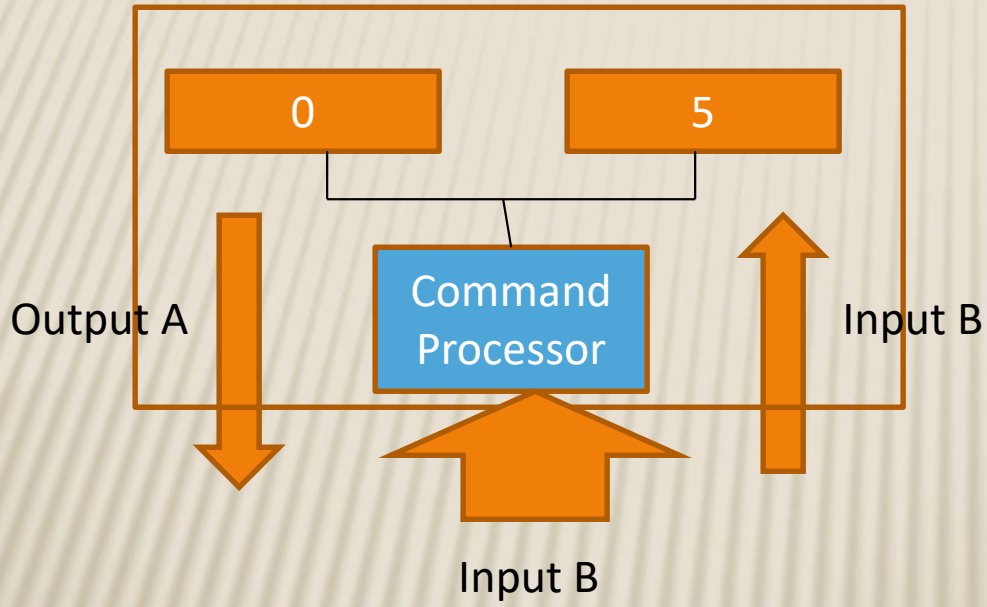
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



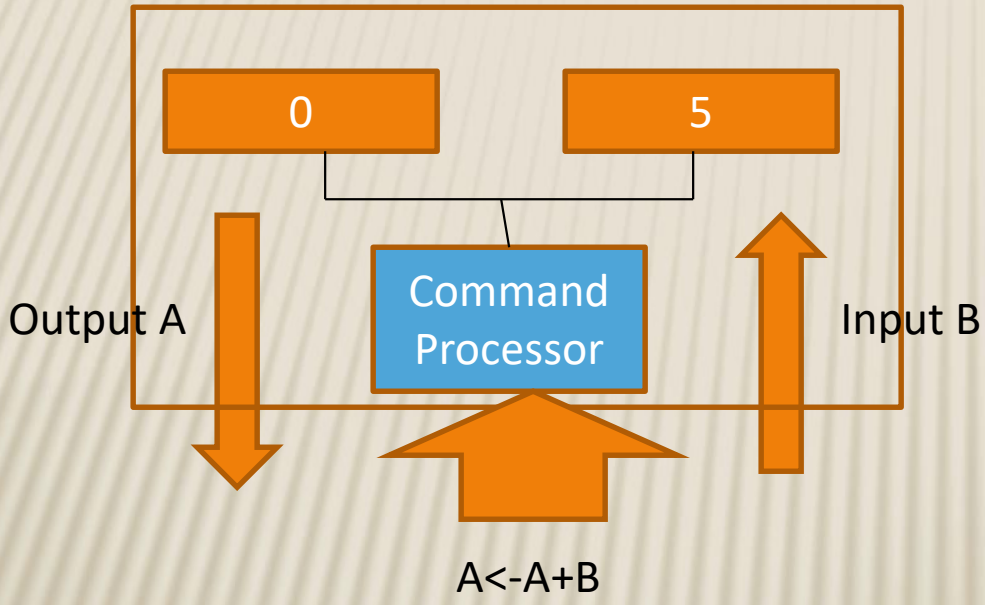
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



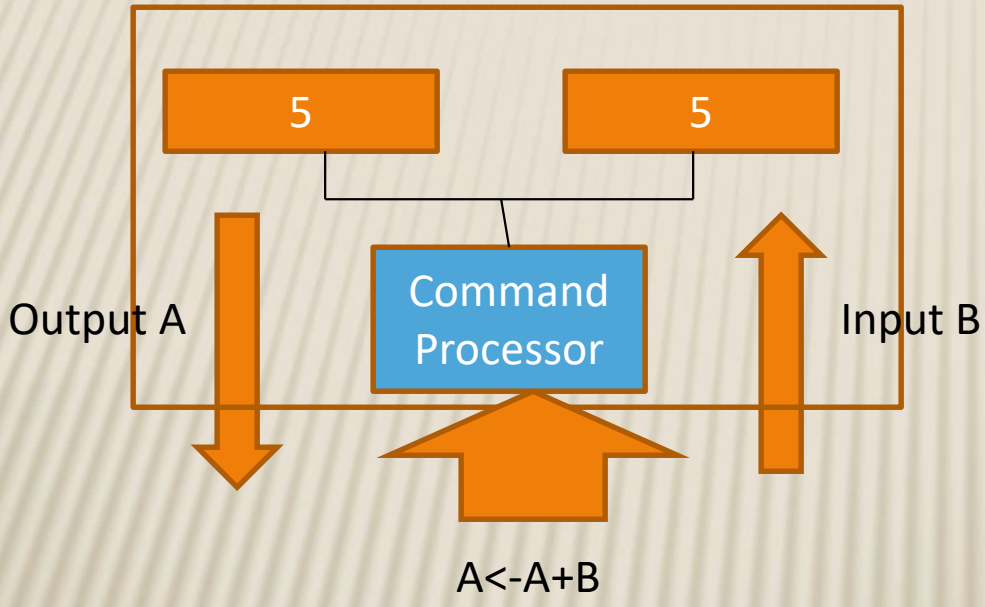
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



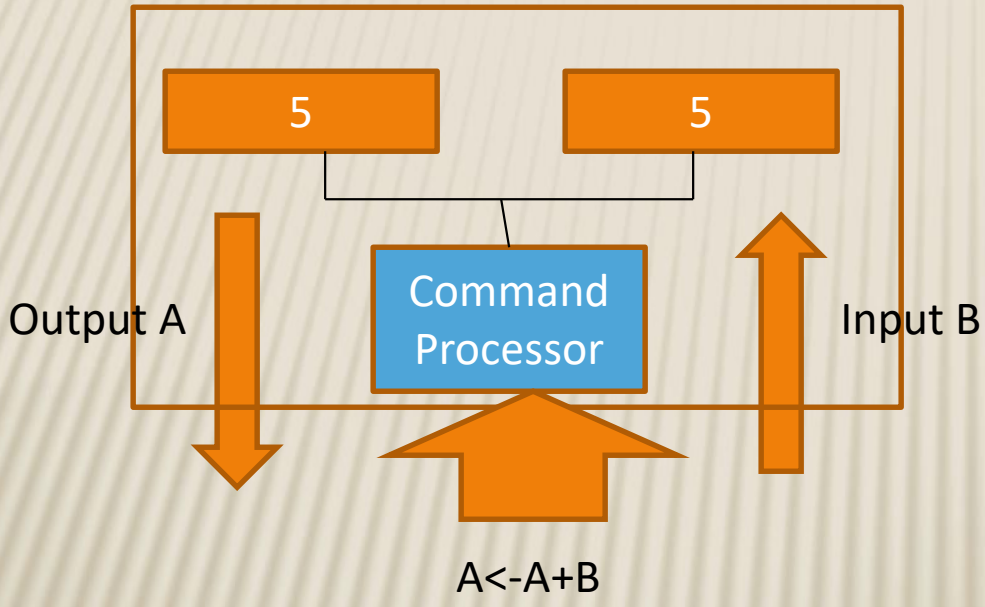
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



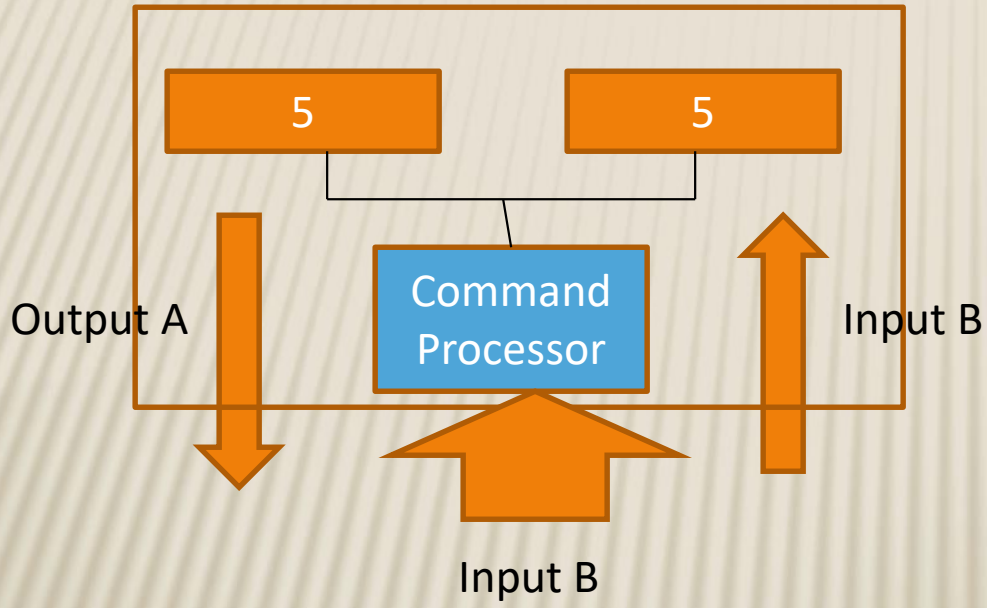
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



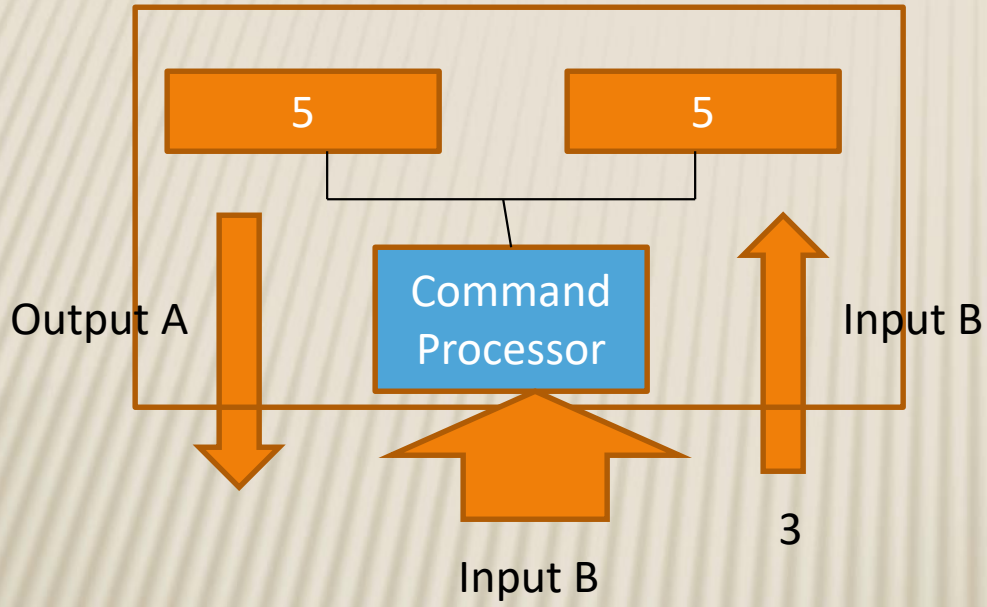
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



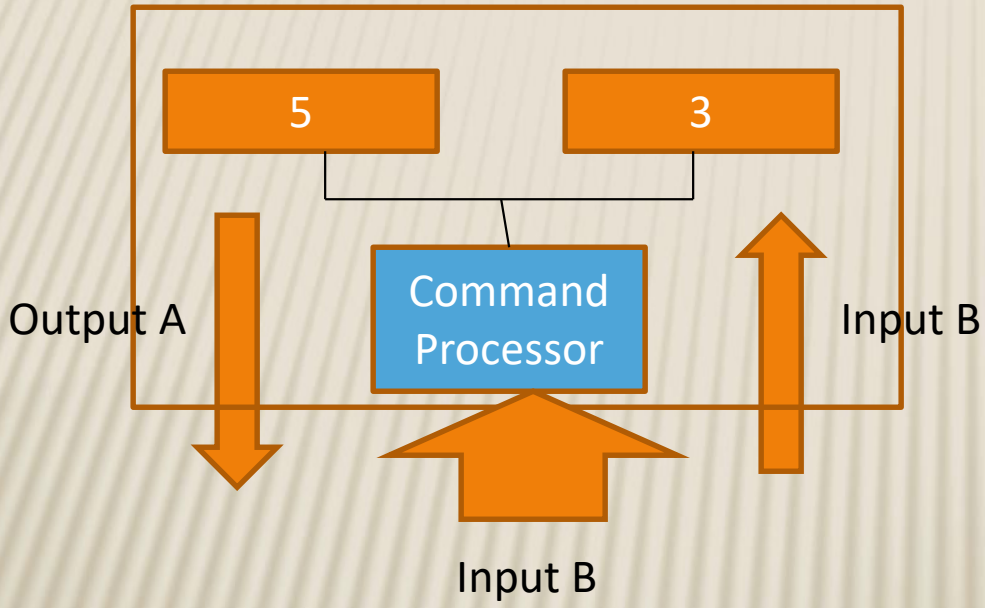
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



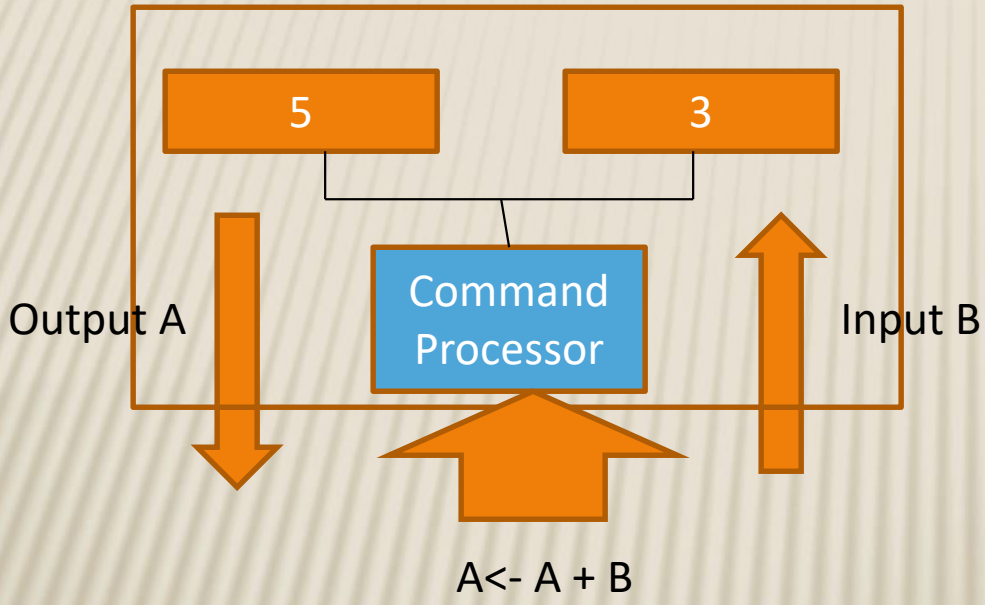
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



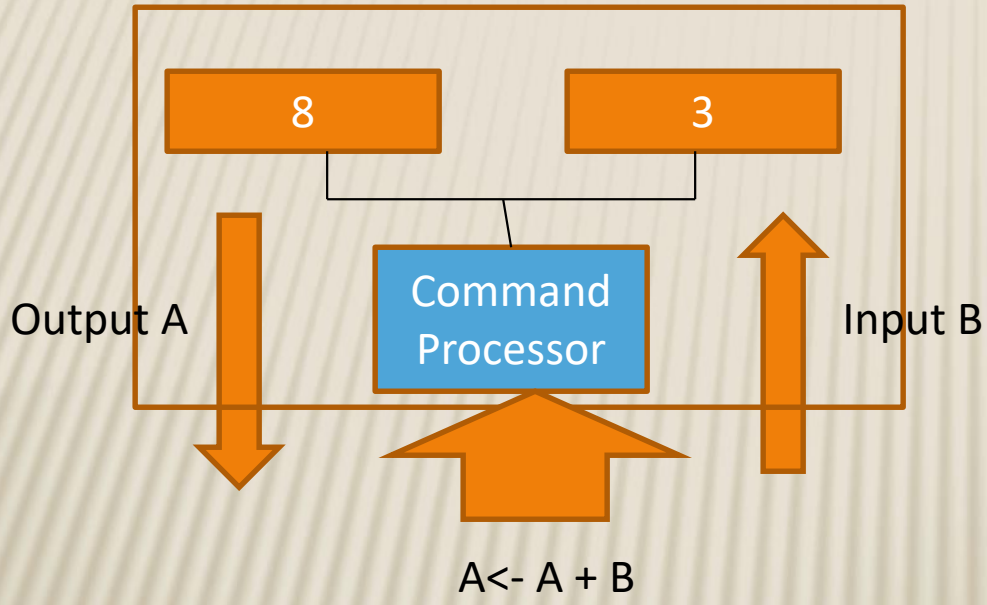
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



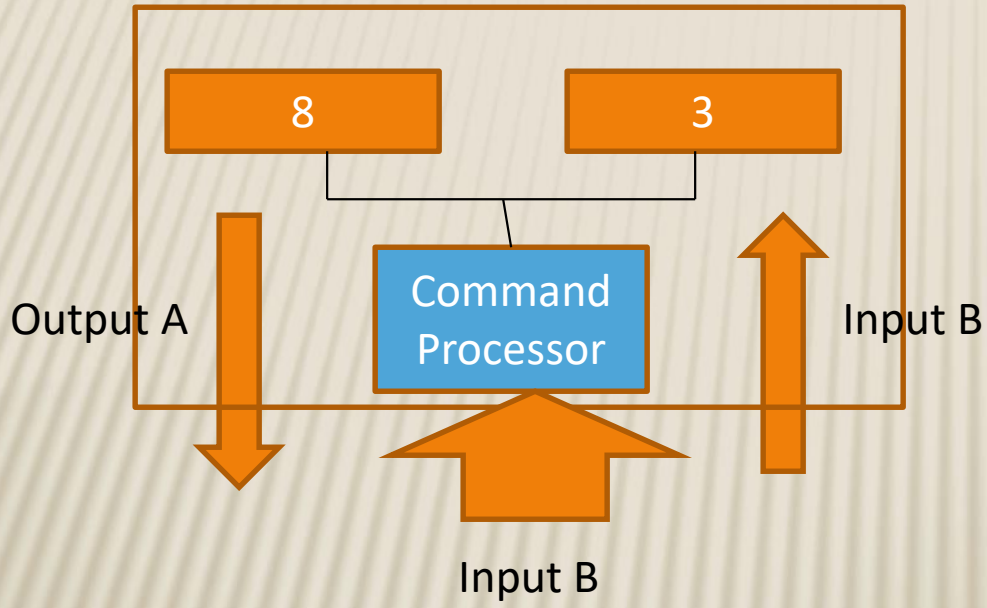
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



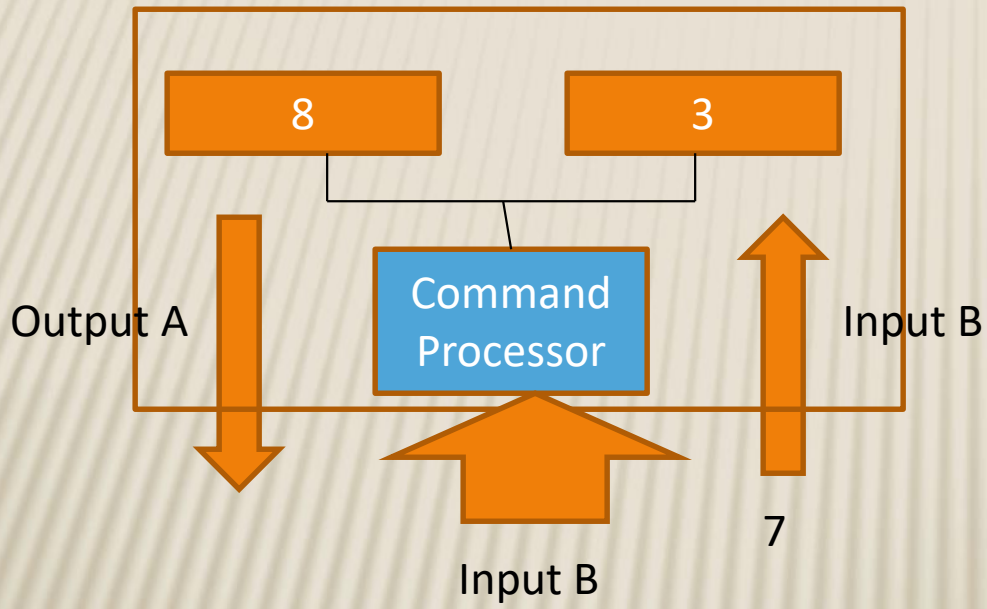
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



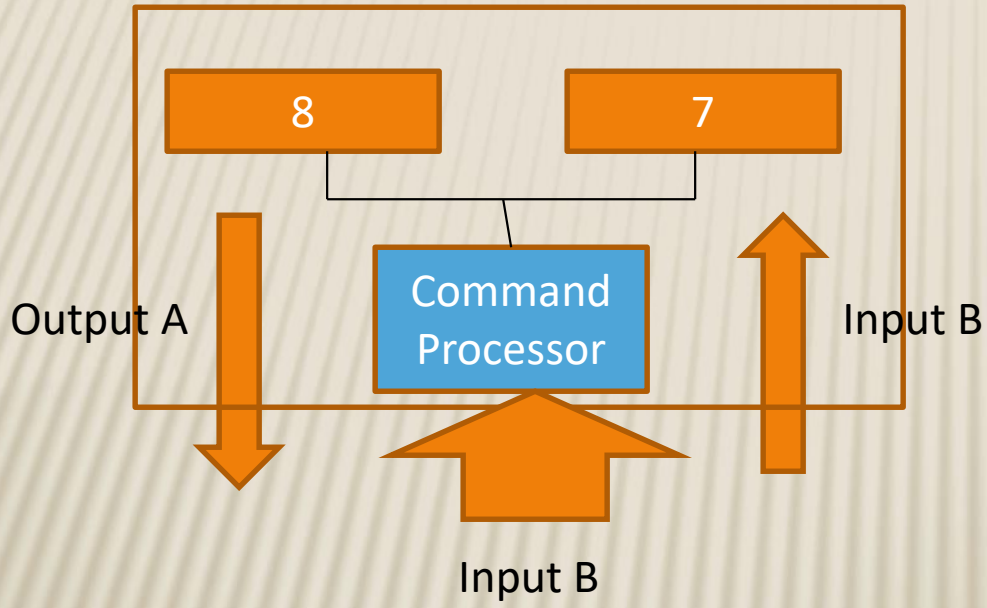
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



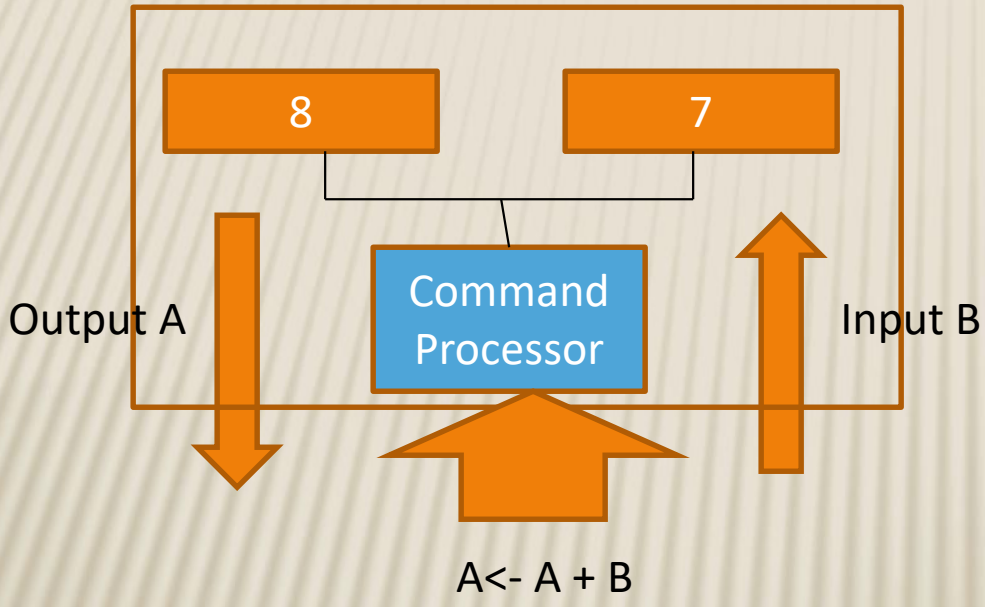
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



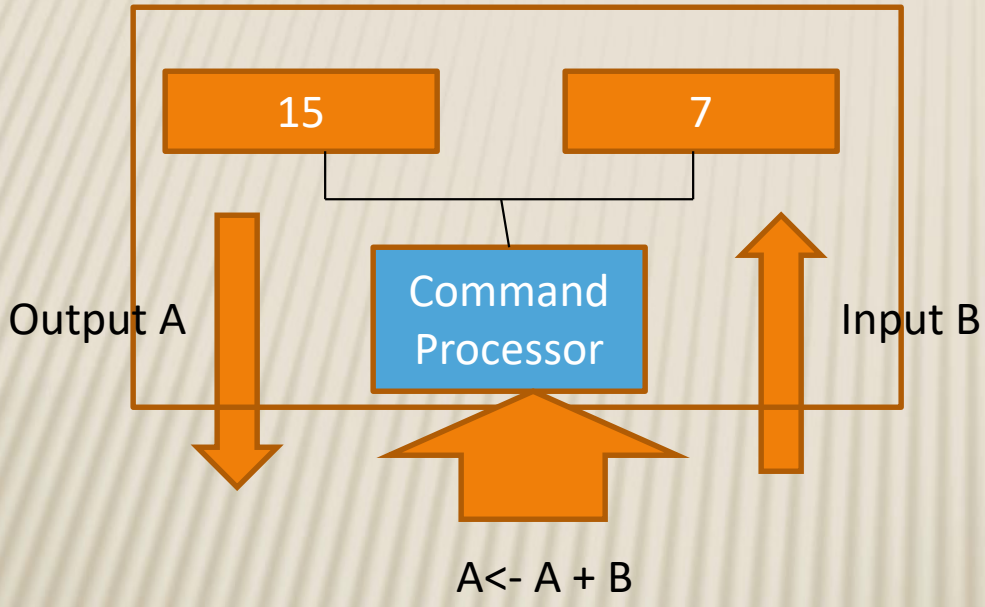
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



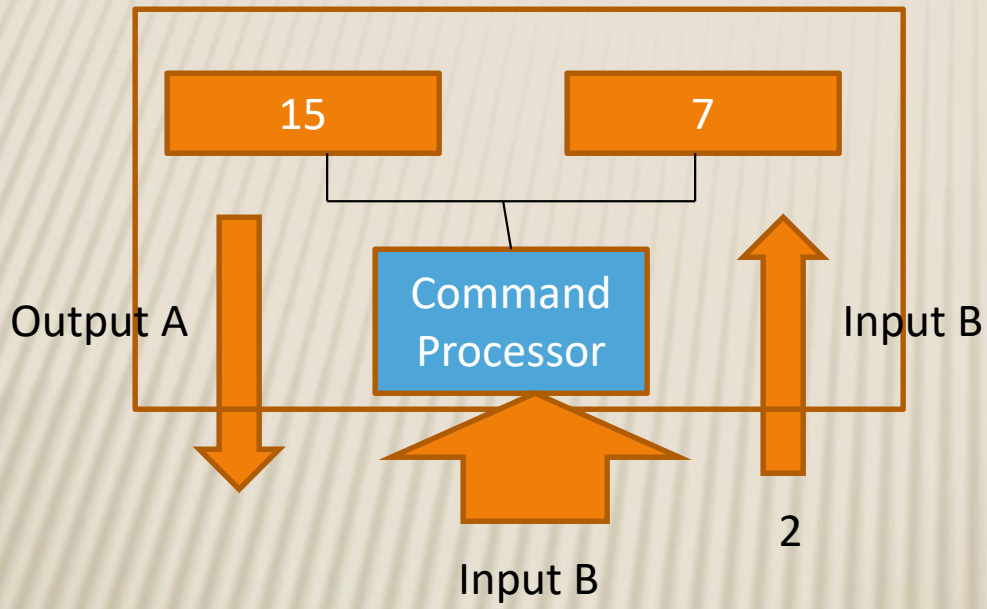
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



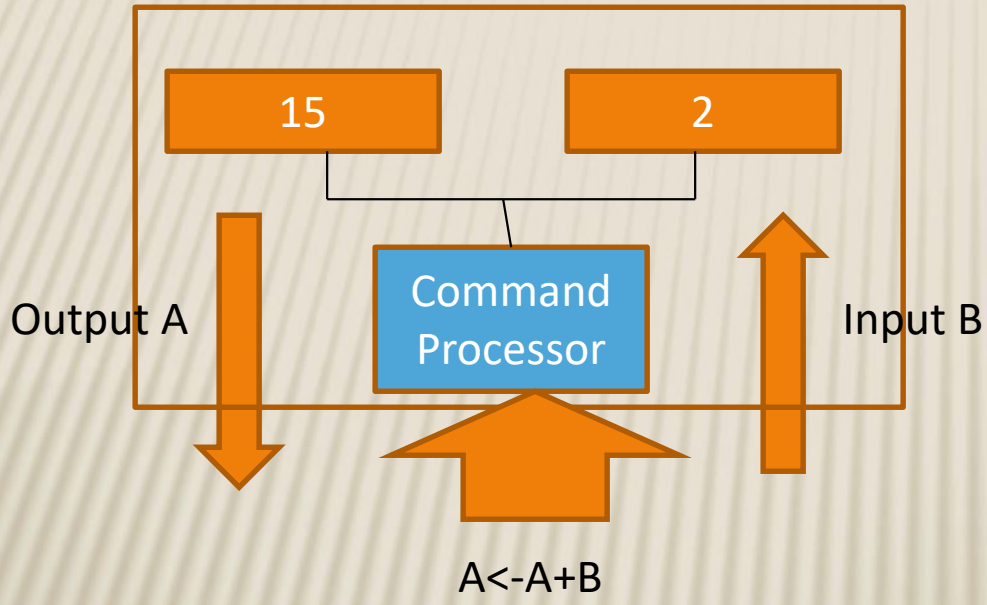
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



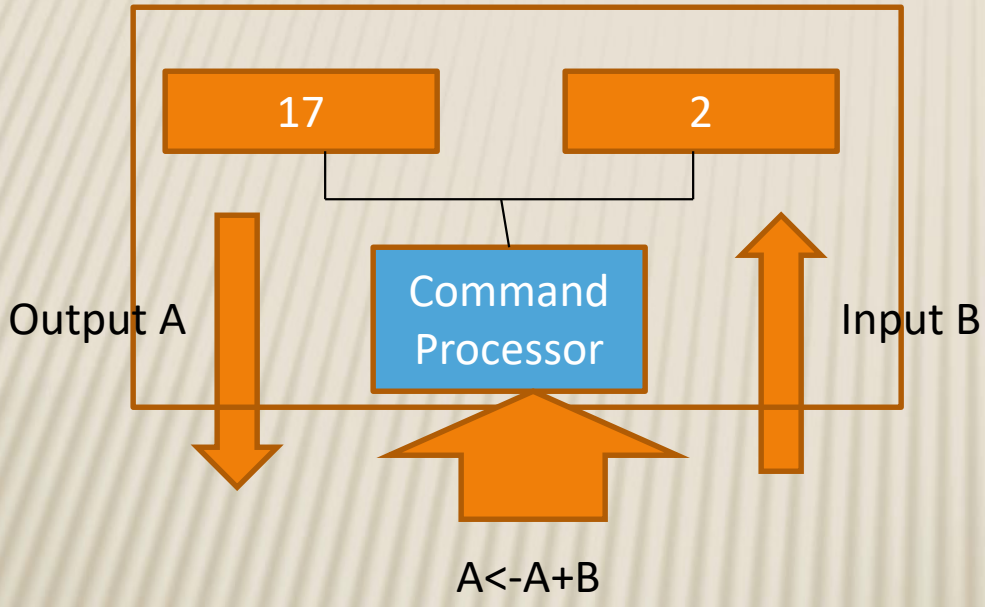
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



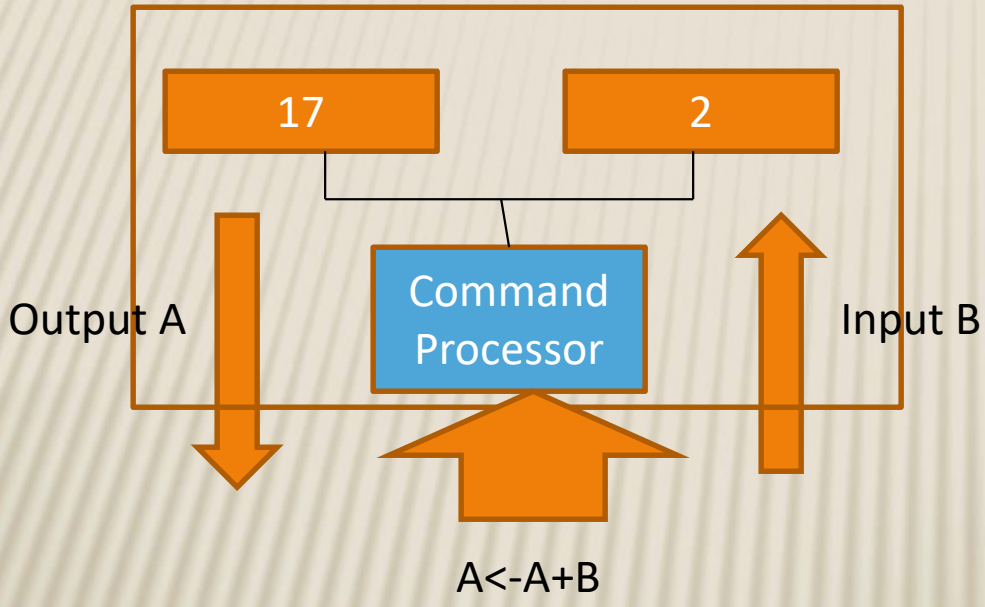
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



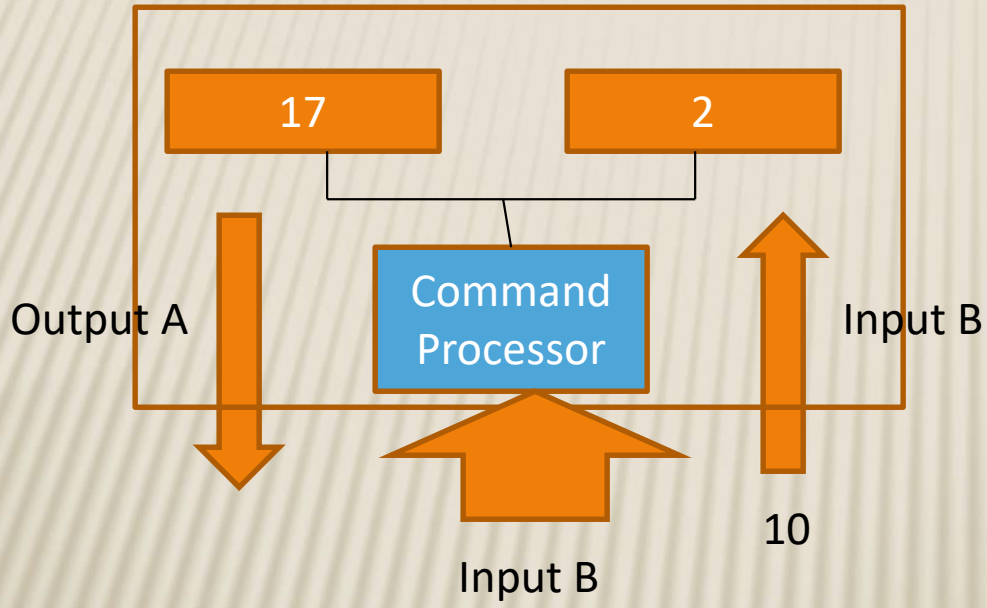
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



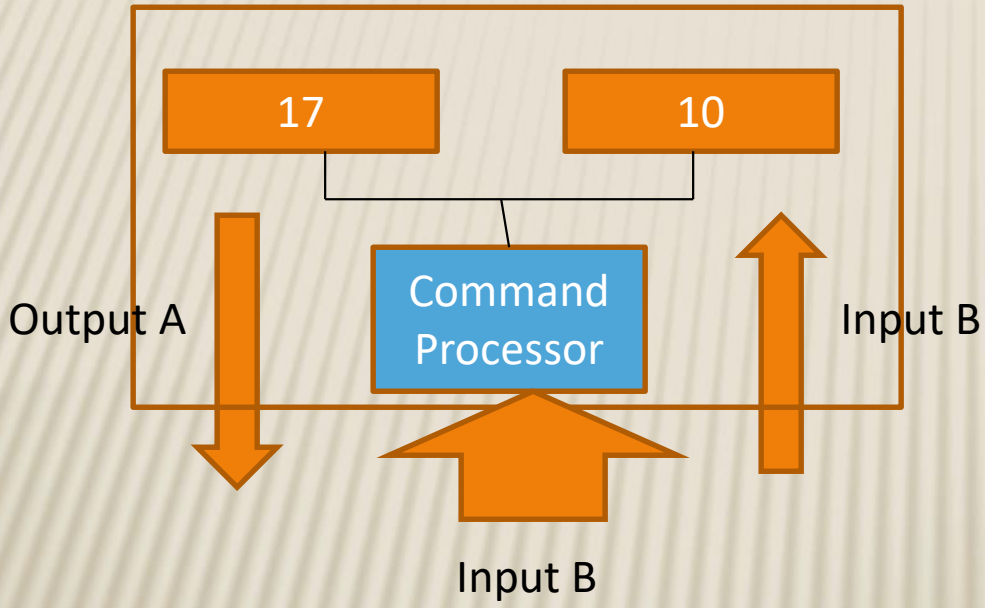
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



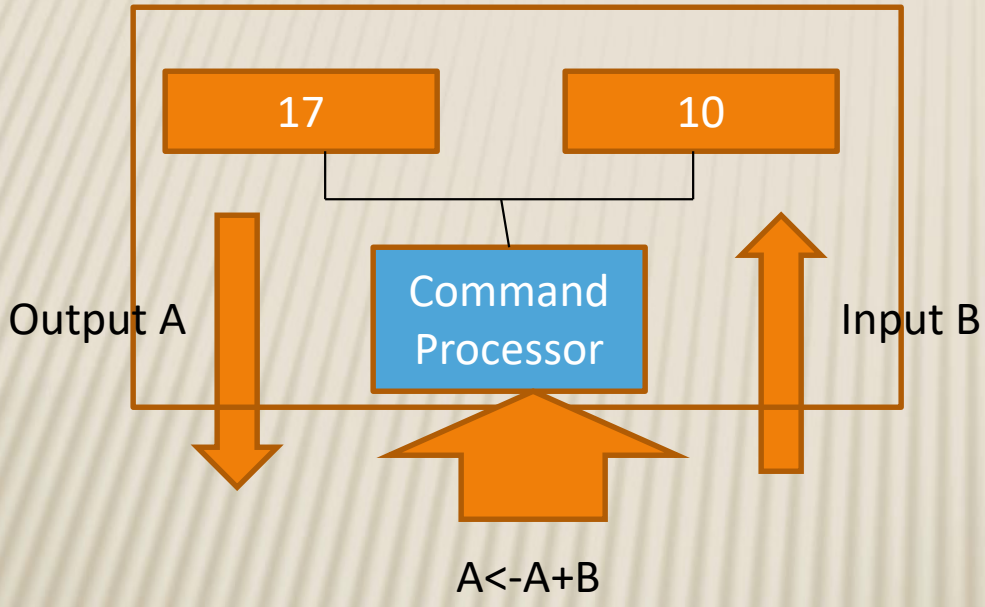
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



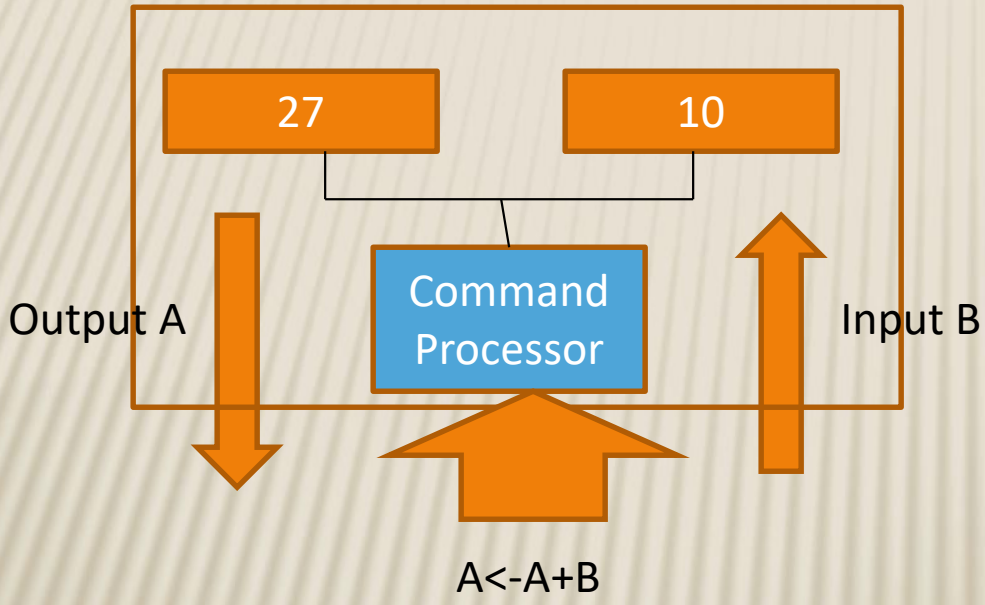
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



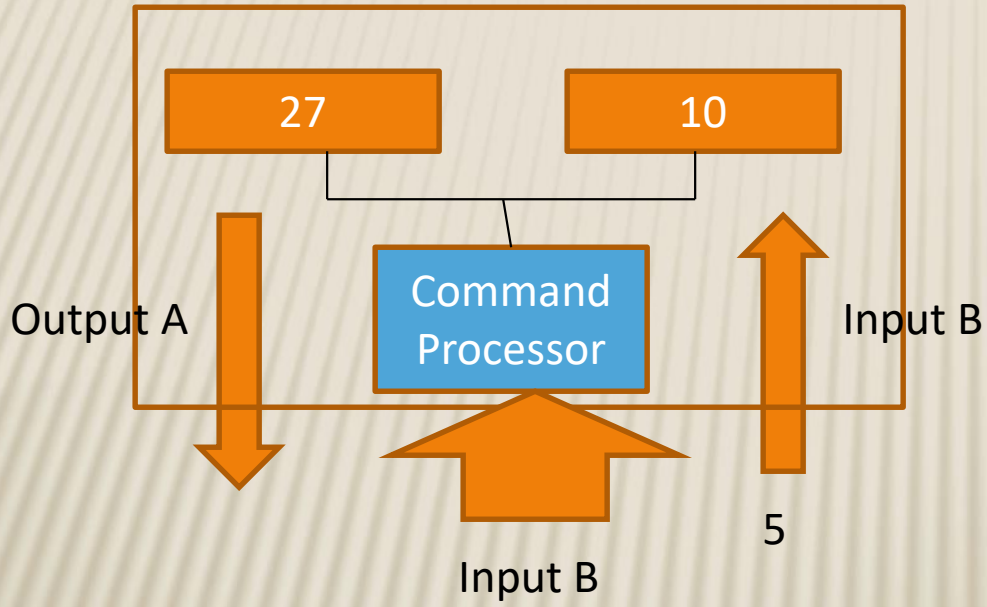
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



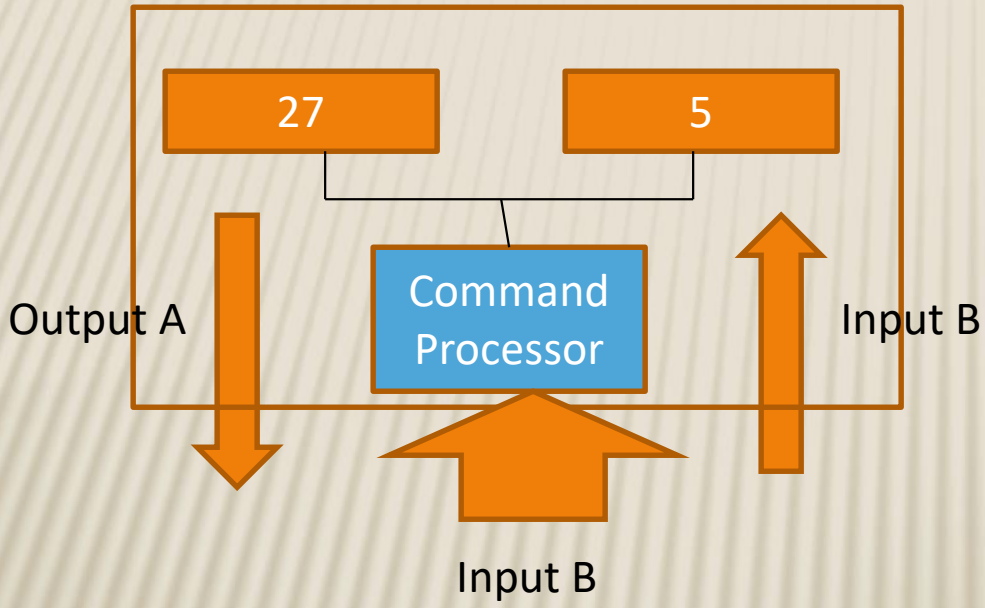
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



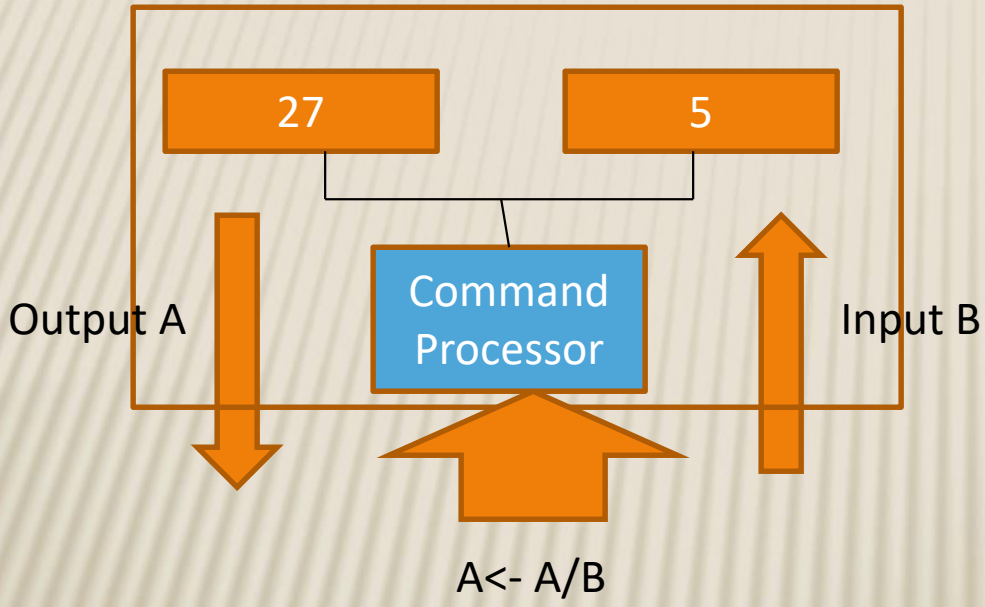
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



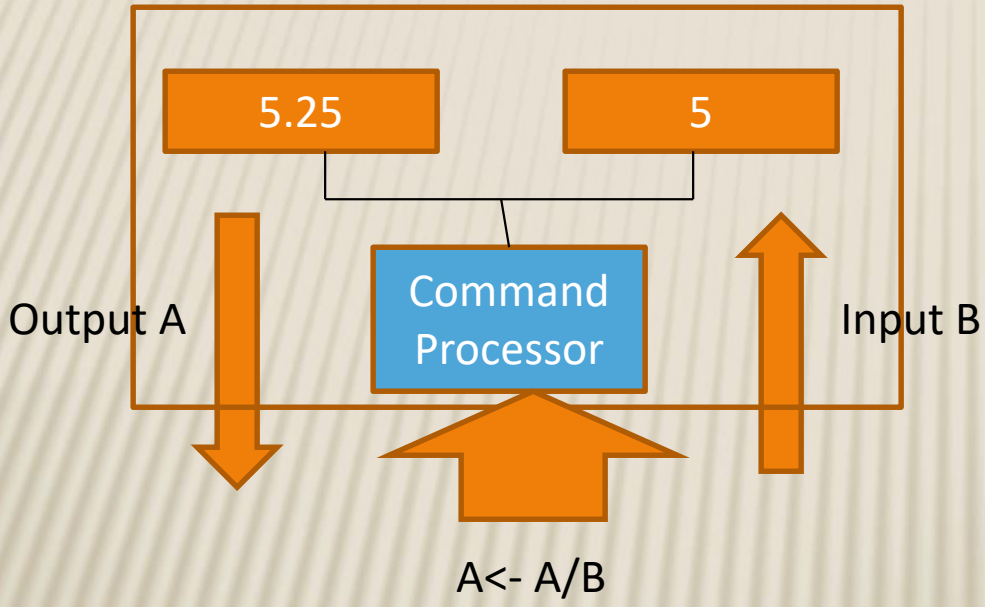
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



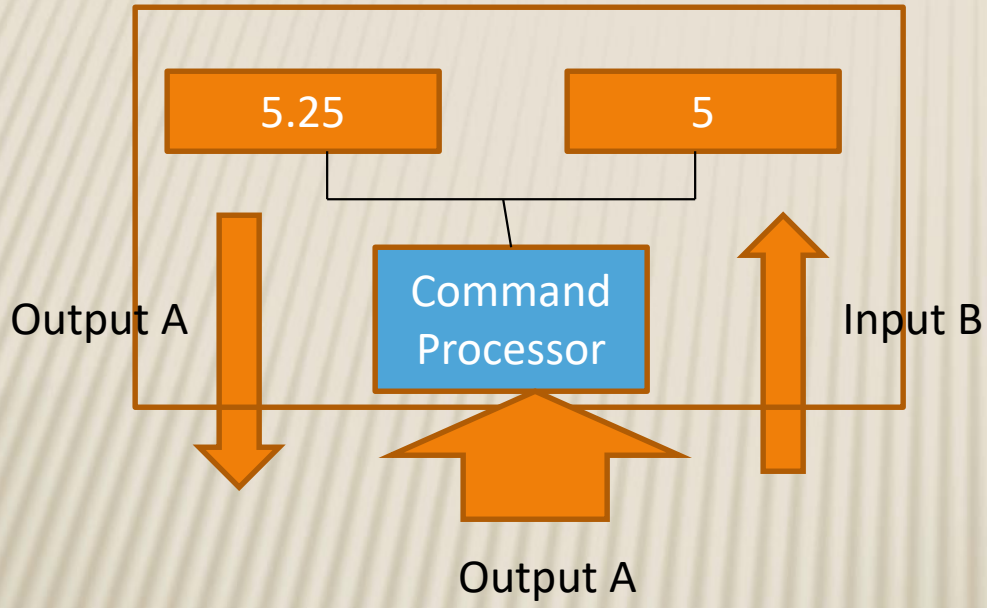
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



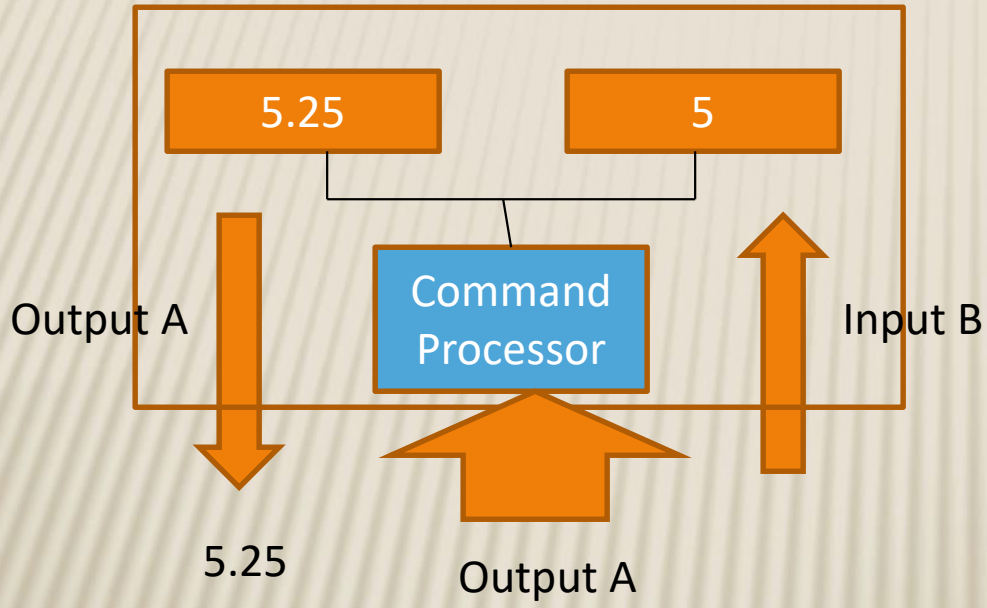
5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



5, 3, 7, 2, 10

P1: CAN WE USE MACHINE A TO FIND THE AVERAGE OF 5 NUMBERS?



5, 3, 7, 2, 10

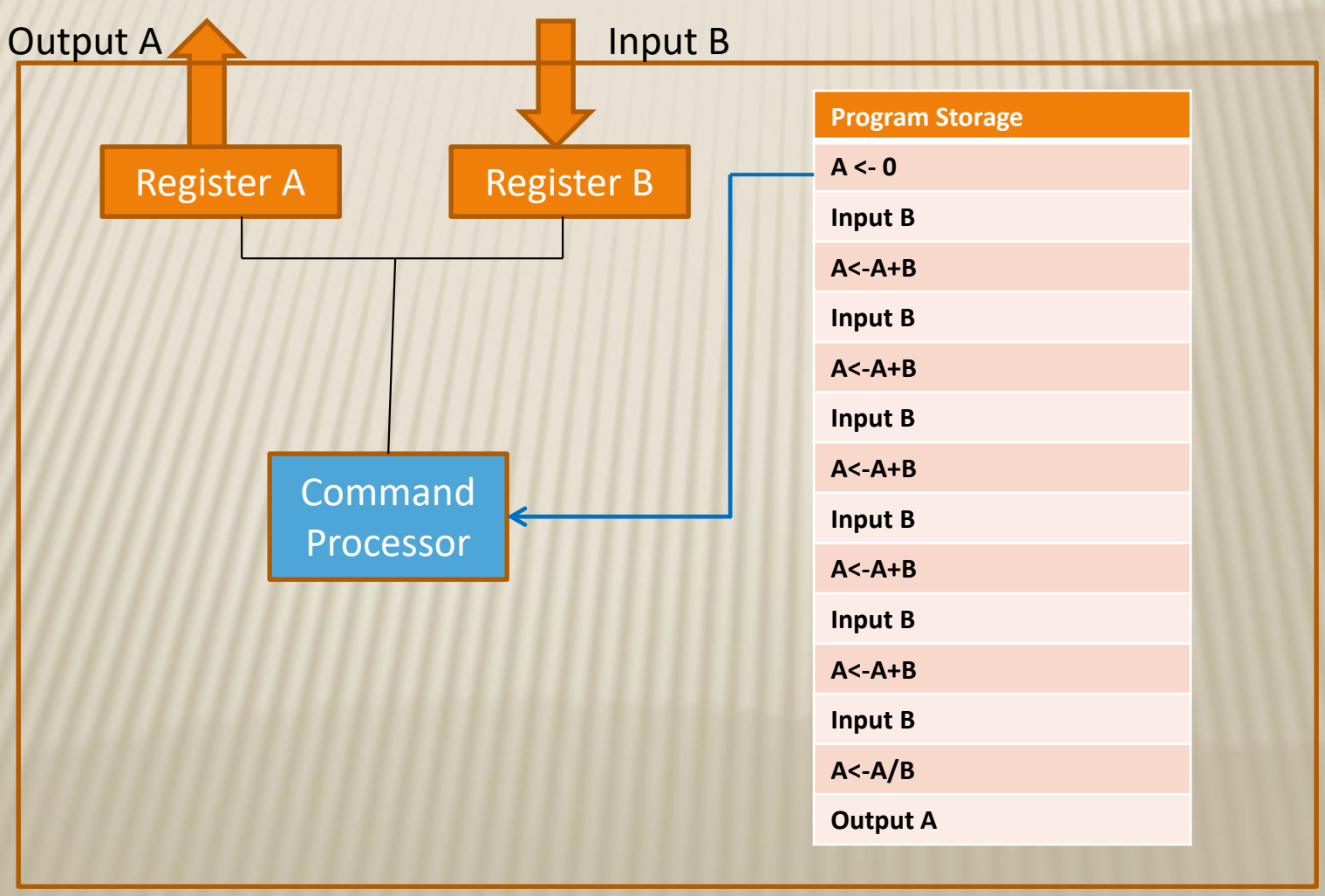
Finally, we got the result!

WHAT IS THE PROBLEM OF MACHINE A?

- You have to re-enter all the command the next time you want to find the average of the 5 numbers!
- Machine A is in fact a calculator

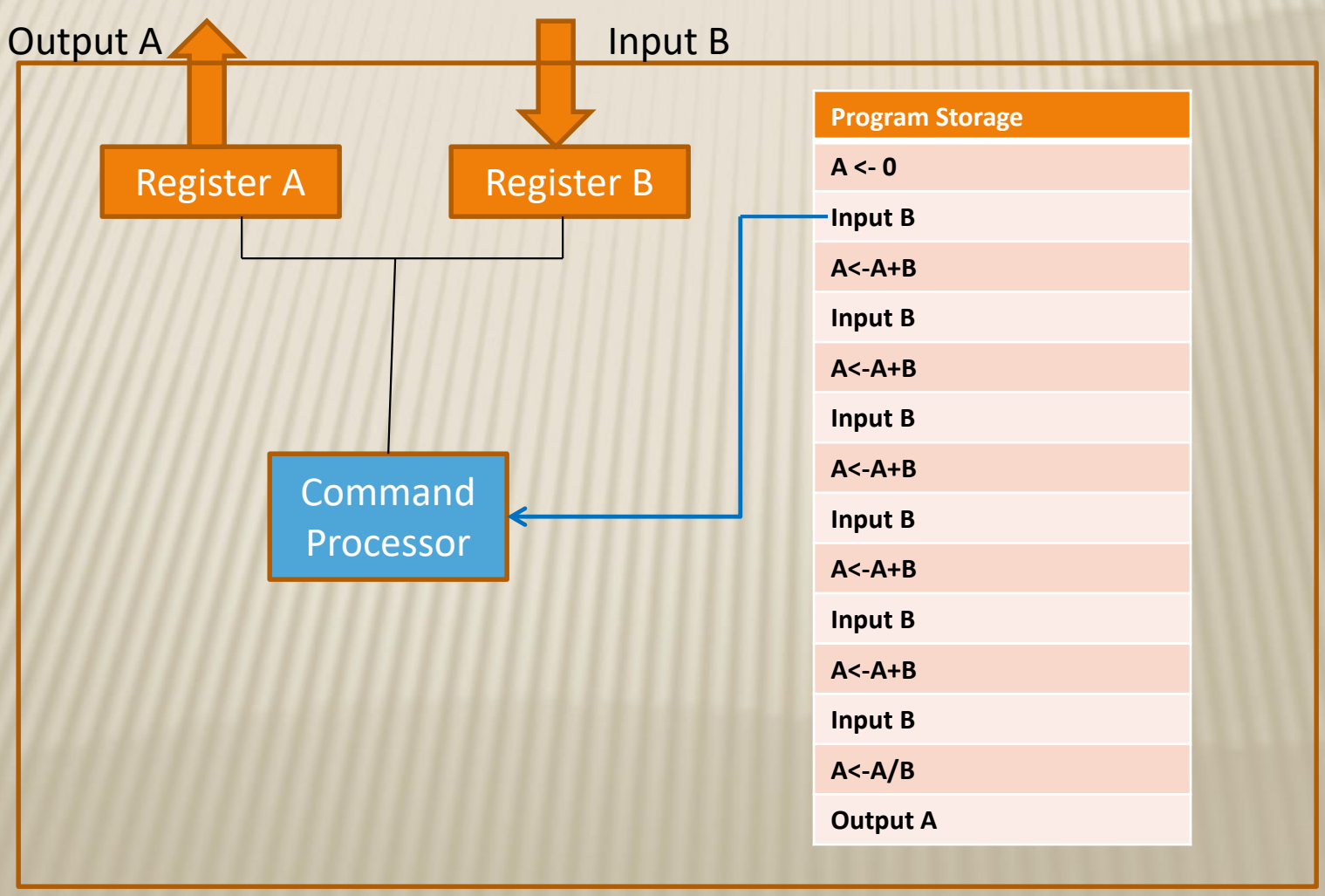
MACHINE B: LET'S STORE THE COMMANDS SOMEWHERE

- We introduce program memory storage to store these commands



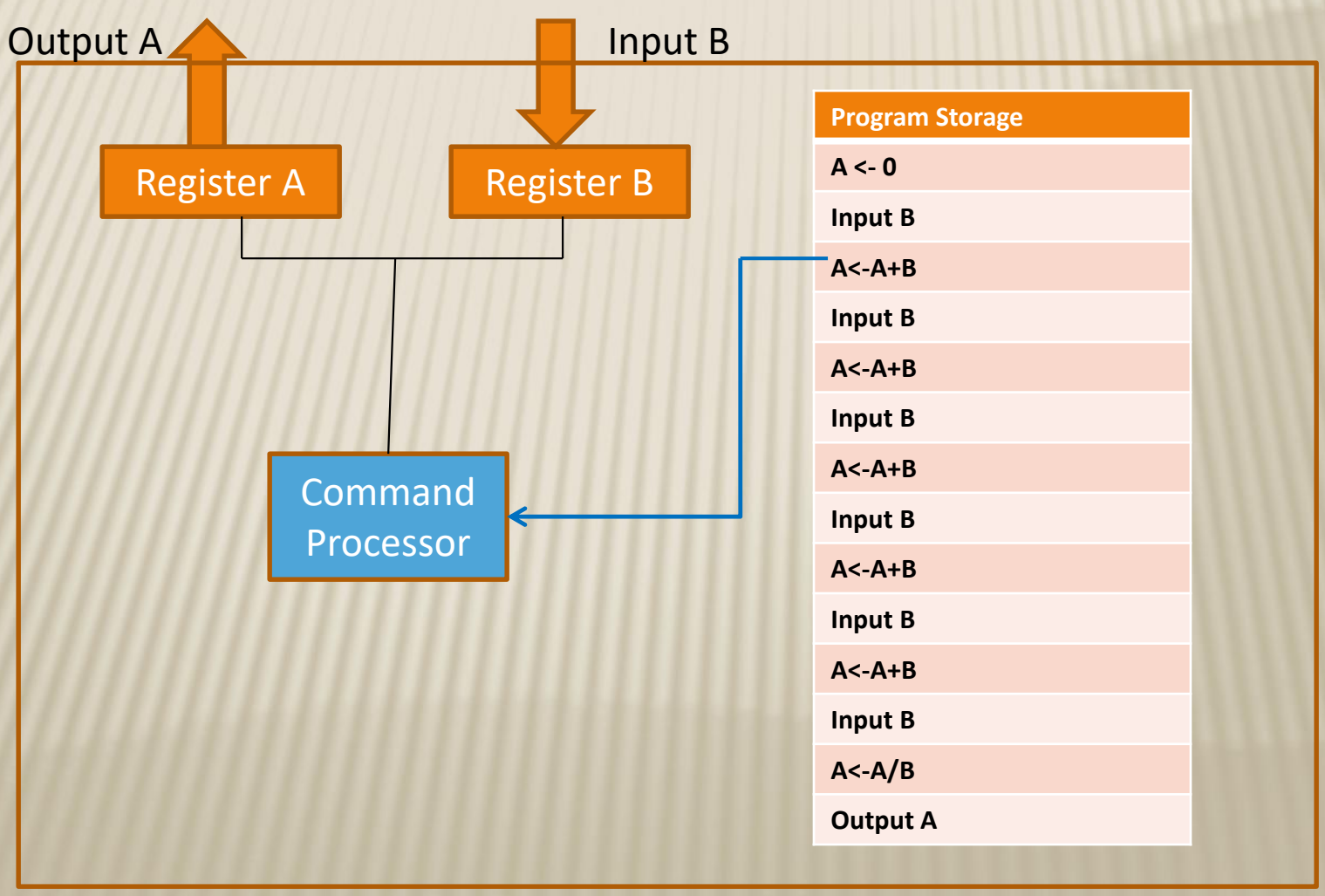
MACHINE B: LET'S STORE THE COMMANDS SOMEWHERE

- We introduce program memory storage to store these commands



MACHINE B: LET'S STORE THE COMMANDS SOMEWHERE

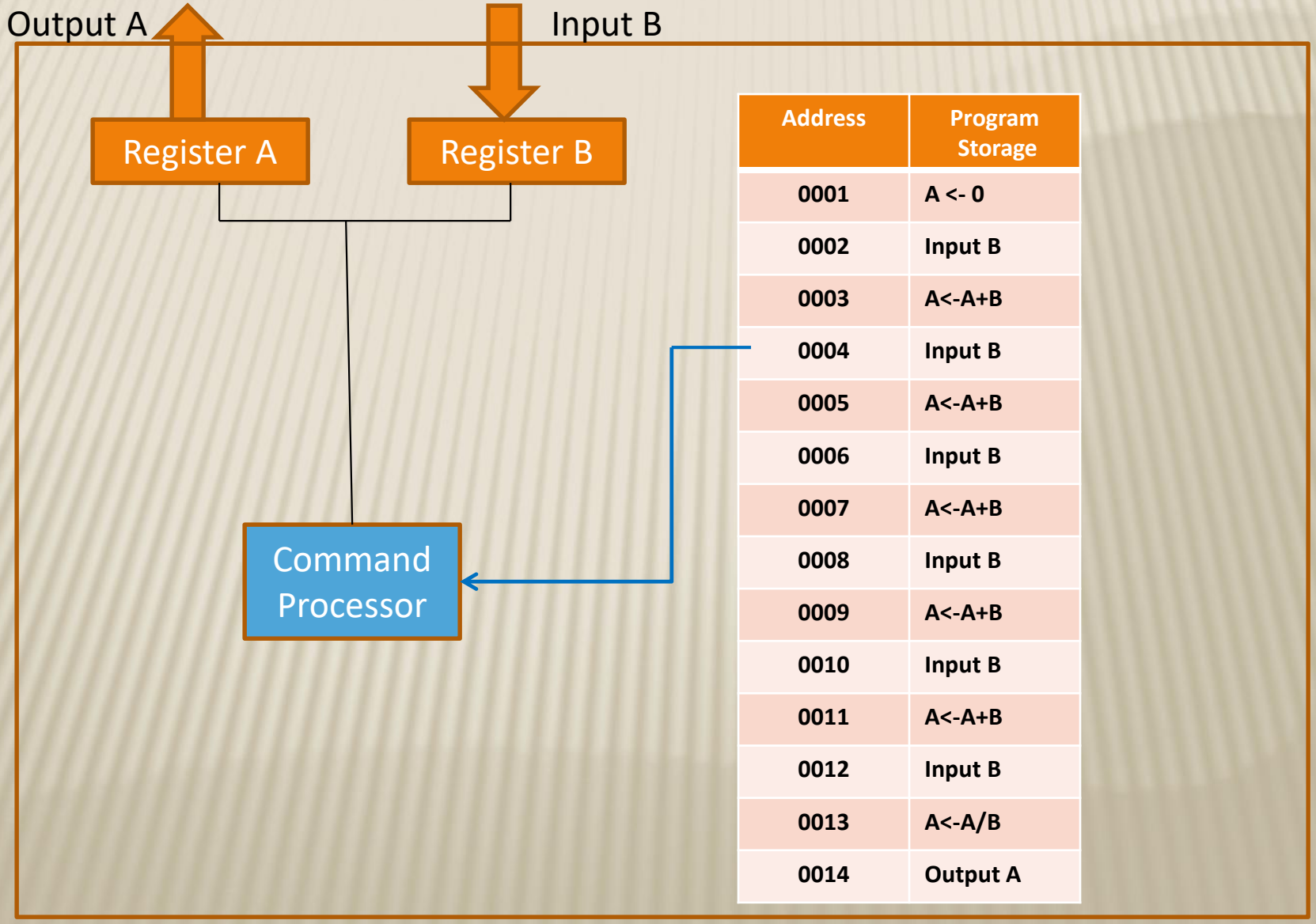
- We introduce program memory storage to store these commands. Automatically execute one after another



MACHINE B ILLUSTRATED WHAT IS PROGRAMMING

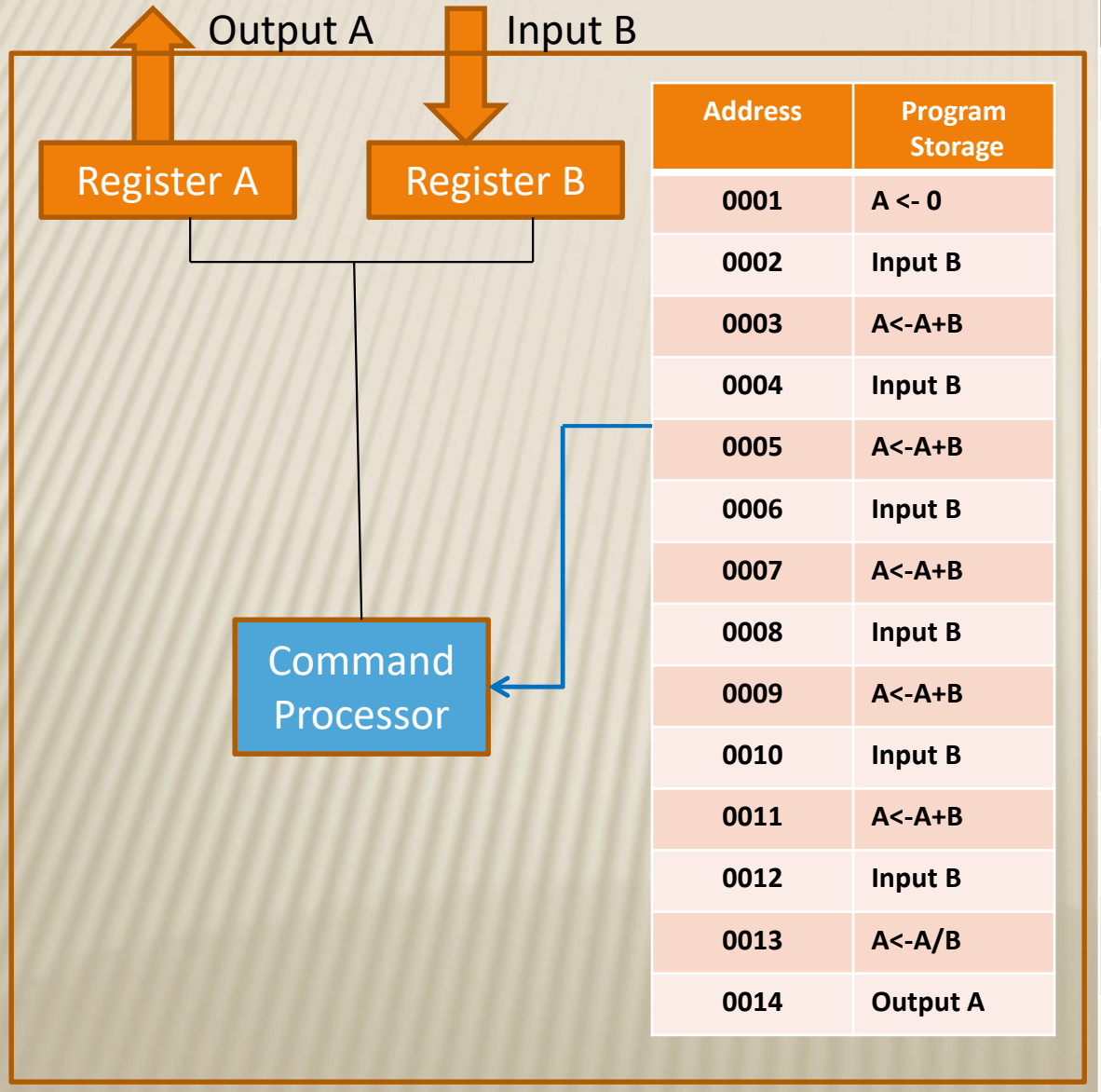
- Essentially, programming involve organizing a set of instructions so that certain tasks can be performed
- Machine B in this case have the same instruction set as A and thus can perform the same task as A
- What if we want to compute the average of **N** numbers where **N** is much larger than 5 (or even not given before hand)?
- Buy lots of memory? Any better solution?

MACHINE C: LET'S MAKE THE INSTRUCTION SET MORE POWERFUL! FIRST LET US ADD IN ADDRESS FOR THE PROGRAM



Address	Program Storage
0001	A <- 0
0002	Input B
0003	A<-A+B
0004	Input B
0005	A<-A+B
0006	Input B
0007	A<-A+B
0008	Input B
0009	A<-A+B
0010	Input B
0011	A<-A+B
0012	Input B
0013	A<-A/B
0014	Output A

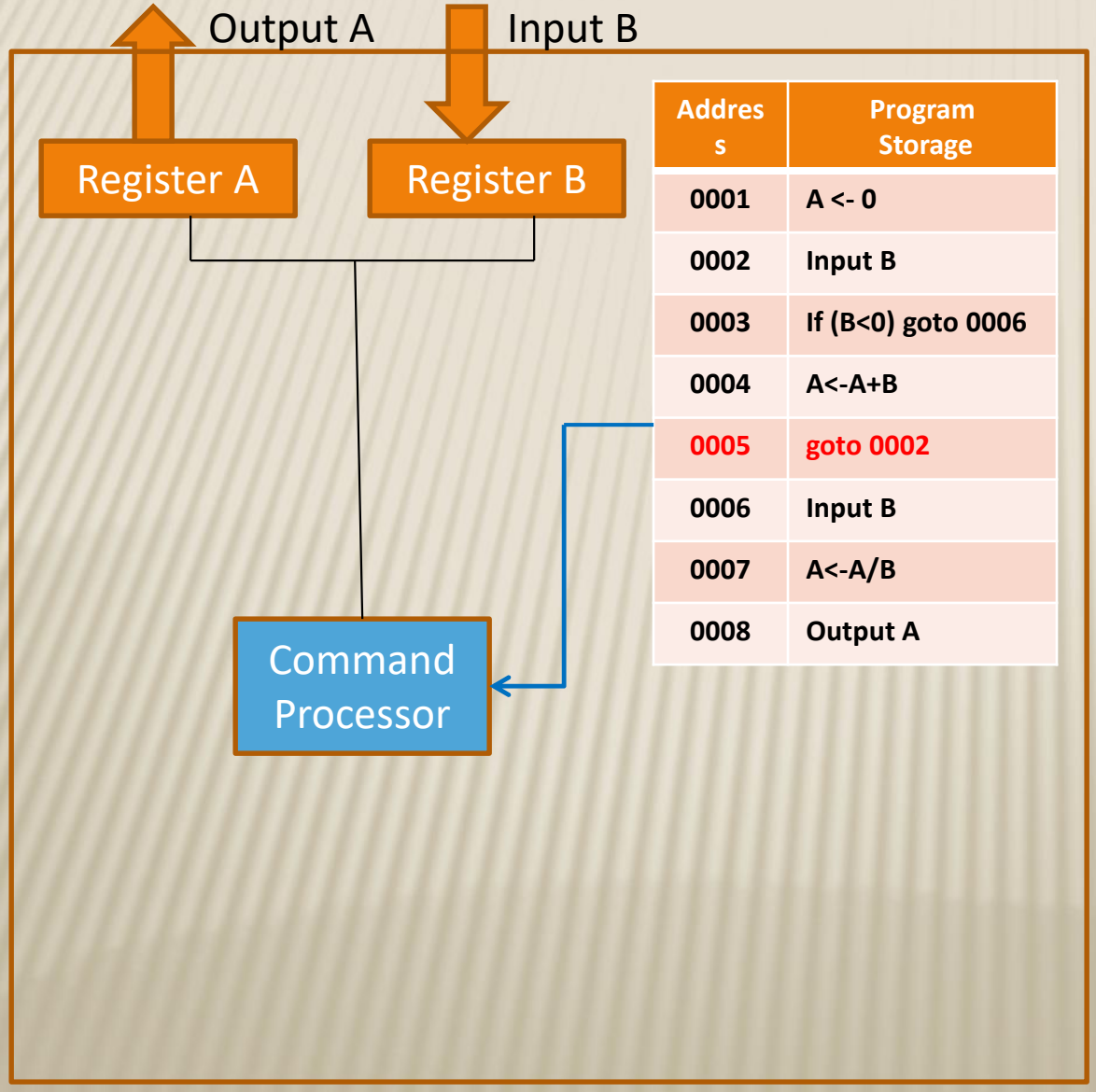
MACHINE C: LET'S MAKE THE INSTRUCTION SET MORE POWERFUL! FIRST LET US ADD IN ADDRESS FOR THE PROGRAM



Address	Program Storage
0001	A <- 0
0002	Input B
0003	A<-A+B
0004	Input B
0005	A<-A+B
0006	Input B
0007	A<-A+B
0008	Input B
0009	A<-A+B
0010	Input B
0011	A<-A+B
0012	Input B
0013	A<-A/B
0014	Output A

Instruction	Effect
InputB	Read a number into Register B
OutputA	Output the number in Register A
A <- v	Set Register A to value v
B <- v	Set Register B to value v
A <- A + B	Sum the values of Register A and B and put the result into A
A <- A/B	Divide the values of Register A and by the value in Register B and put the result into A
If (B=v) goto *add*	If Register B= v, then jump to instruction at address *add*
If (B<v) goto *add*	If Register B < v, then jump to instruction at address *add*
If (B>v) goto *add*	If Register B > v, then jump to instruction at address *add*

MACHINE C: LET'S MAKE THE INSTRUCTION SET MORE POWERFUL! FIRST LET US ADD IN ADDRESS FOR THE PROGRAM



Address	Program Storage
0001	A <- 0
0002	Input B
0003	If (B<0) goto 0006
0004	A<-A+B
0005	goto 0002
0006	Input B
0007	A<-A/B
0008	Output A

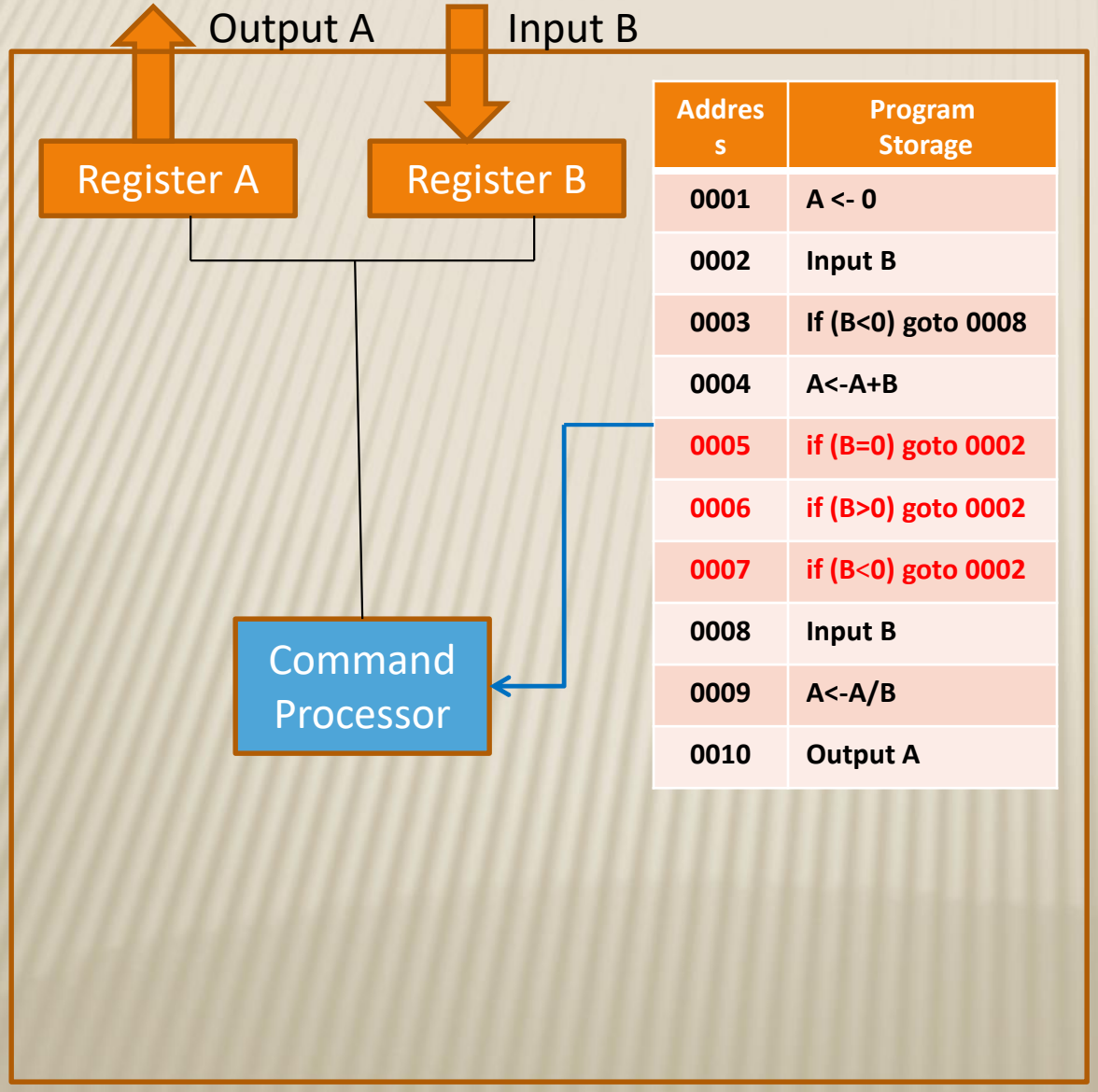
Input: 5, 3, 7, 2, 10, 9, -1, 6

Notice how the program get smaller and more flexible at the expense of the added complexity

Instruction at 0005 is not really valid, how do we implement it using the valid instruction format?

Instruction	Effect
If (B=v) goto *add*	If Register B= v, then jump to instruction at address *add*
If (B<v) goto *add*	If Register B < v, then jump to instruction at address *add*
If (B>v) goto *add*	If Register B > v, then jump to instruction at address *add*

MACHINE C: LET'S MAKE THE INSTRUCTION SET MORE POWERFUL! FIRST LET US ADD IN ADDRESS FOR THE PROGRAM



Address	Program Storage
0001	A <- 0
0002	Input B
0003	If (B<0) goto 0008
0004	A<-A+B
0005	if (B=0) goto 0002
0006	if (B>0) goto 0002
0007	if (B<0) goto 0002
0008	Input B
0009	A<-A/B
0010	Output A

Instruction at 0005 is not really valid, how do we implement it using the valid instruction format?

Here statement 0005,0006,008 in fact is **equivalent** to

goto 0002

Instruction	Effect
	. . .
If (B=v) goto *add*	If Register B= v, then jump to instruction at address *add*
If (B<v) goto *add*	If Register B < v, then jump to instruction at address *add*
If (B>v) goto *add*	If Register B > v, then jump to instruction at address *add*

ASIDE FROM AVOIDING THE USE OF ADDRESS…

- while, if-then-else, for statement in C also enforce certain discipline in program logic

Address	Program Storage
0001	
0002	goto 0005
0003	
0004	
0005	
0006	goto 0001
0007	
0008	goto 0004
0009	
0010	goto 0007

Uncontrolled use of goto lead to spaghetti code

Program Storage
……
while (b<0) do
{ if (a >0) then
{ ……
for (x = 0; x < 10; x++)
{ …..
……
} //endfor
} else
{
} // endif
}//endwhile

Arrows seldom cross each other here !

CAN MACHINE C REALLY COMPUTE EVERYTHING?

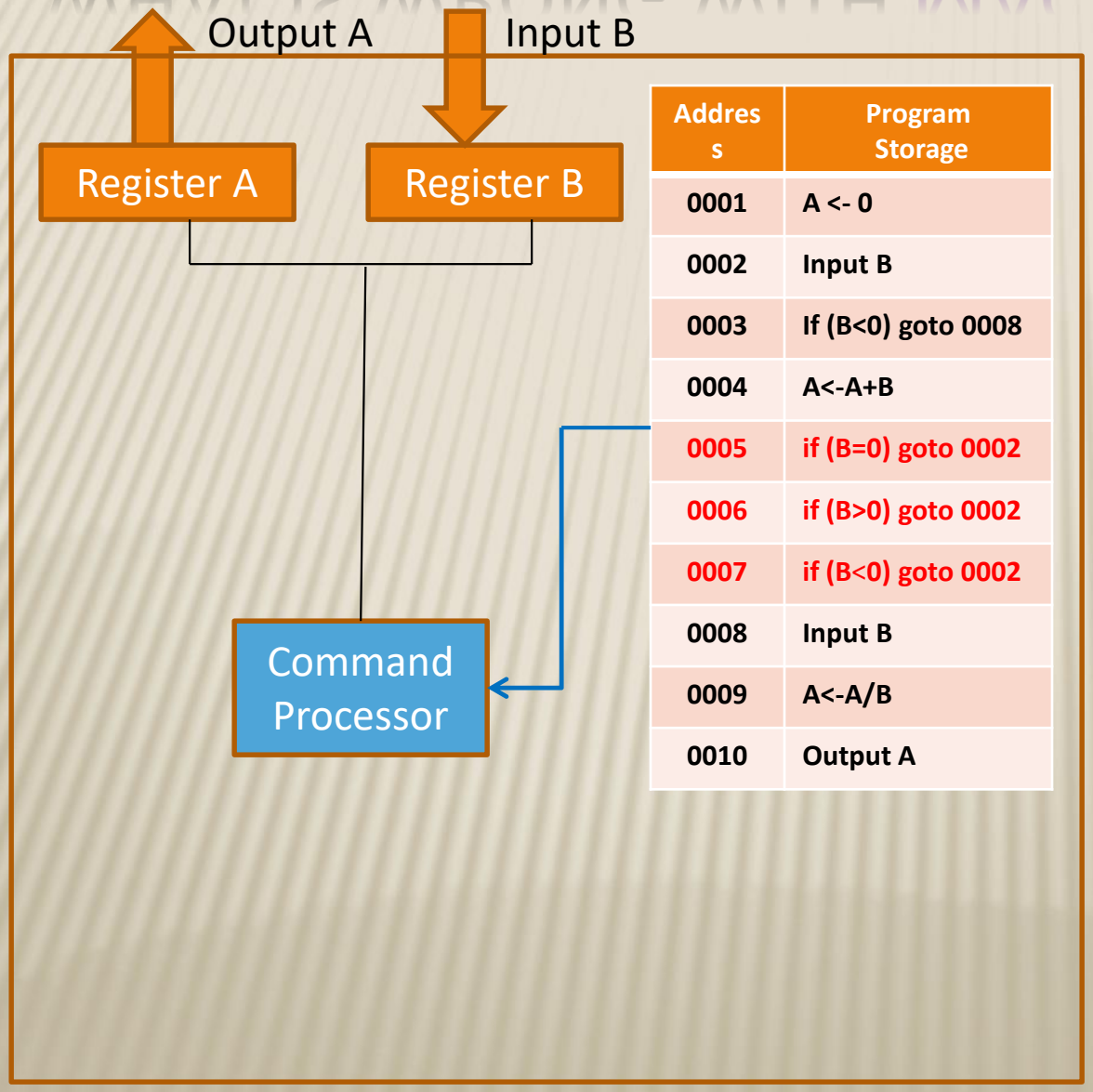
- Recall our input data to Machine C
  - 5, 3, 7, 2, 10, 9, -1, 6

Why do we have to tell the machine how many numbers are there? Can't Machine C keep count of how many numbers are there? Are we cheating somehow?

Problem 1a: Can we use Machine C to compute the average without telling the machine how many numbers are there?

Problem 1b: Can we use Machine C to compute the median instead of the average ?

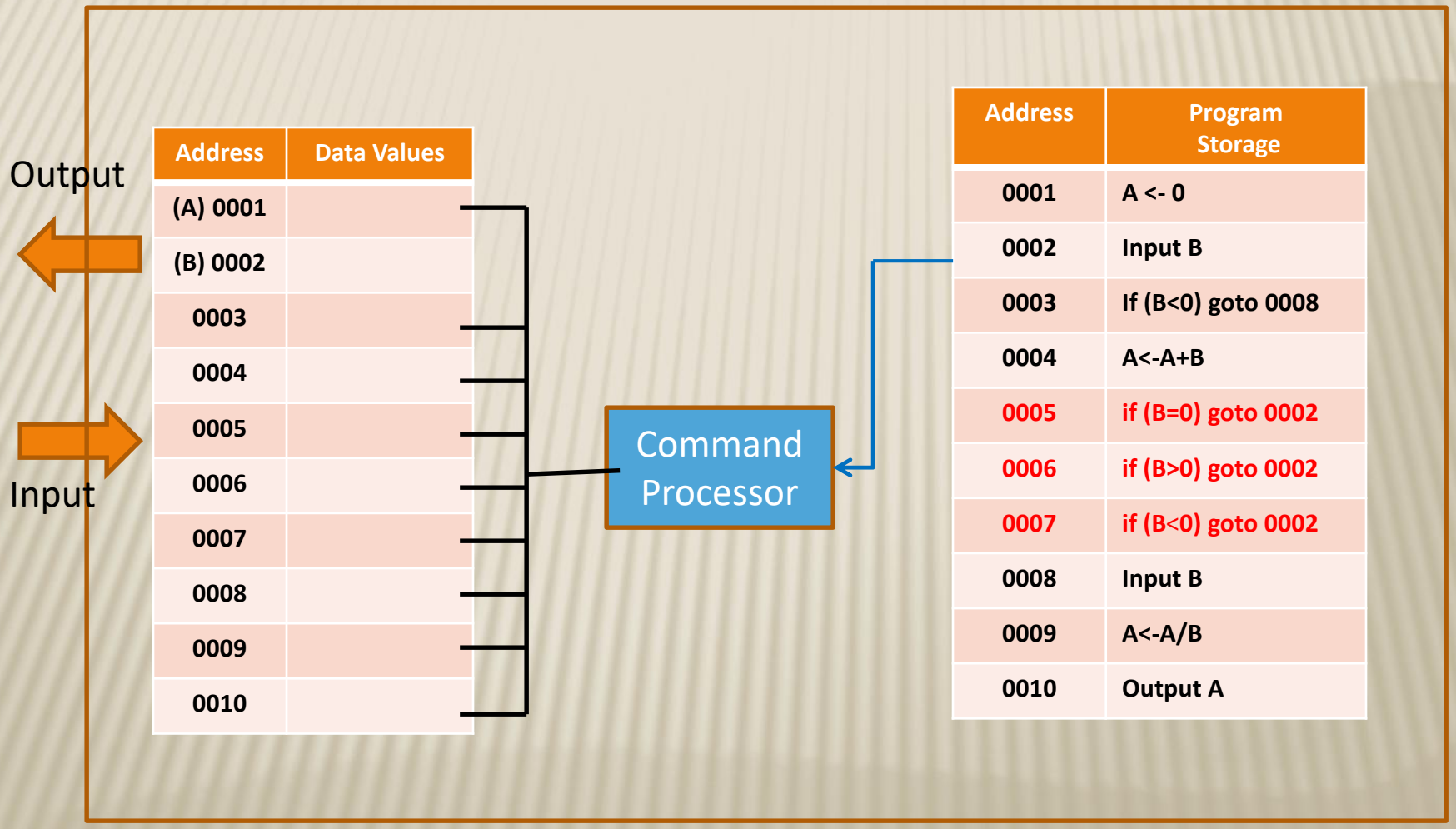
WHAT IS WRONG WITH MACHINE C?



Address	Program Storage
0001	A <- 0
0002	Input B
0003	If (B<0) goto 0008
0004	A<-A+B
0005	if (B=0) goto 0002
0006	if (B>0) goto 0002
0007	if (B<0) goto 0002
0008	Input B
0009	A<-A/B
0010	Output A

- We essential do not have memory storage for data!

MACHINE D: WE REPLACE REGISTERS WITH MEMORY FOR DATA



INSTRUCTION SET FOR MACHINE D

Instruction Type	Instruction	Effect
I/O Statement	Input addr	Read a number into data location addr
I/O Statement	Output addr	Output the number at data location addr
Assignment Statement	addr <- v	Set location addr to value v
Assignment/Data Operator	addr <- v(addr1) + v(addr2)	Sum the values of at location addr1 and addr2 and put the result at location addr
Assignment/Data Operator	addr <- v(addr1)/v(addr2)	Divide the values at location addr1 by value at addr2 and put the result at location addr
Control Statement	If (v(addr)= v) goto *add1*	If value at location addr = v, then jump to instruction at address *add1*
Control Statement	If (v(addr)<v) goto *add1*	If value at location < v, then jump to instruction at address *add1*
Control Statement	If (v(addr)>v) goto *add1*	If value at location > v, then jump to instruction at address *add1*

As long as you know the syntax for using the four instruction types above, you can learn any programming language!

INSTRUCTION SET FOR MACHINE D

Instruction Type	Instruction	Effect
I/O Statement	Input addr	Read a number into data location addr
I/O Statement	Output addr	Output the number at data location addr
Assignment Statement	addr <- v	Set location addr to value v
Assignment/Data Operator	addr <- v(addr1) + v(addr2)	Sum the values of at location addr1 and addr2 and put the result at location addr
Assignment/Data Operator	addr <- v(addr1)/v(addr2)	Divide the values at location addr1 by value at addr2 and put the result at location addr
Control Statement	If (v(addr)= v) goto *add1*	If value at location addr = v, then jump to instruction at address *add1*
Control Statement	If (v(addr)<v) goto *add1*	If value at location < v, then jump to instruction at address *add1*
Control Statement	If (v(addr)>v) goto *add1*	If value at location > v, then jump to instruction at address *add1*

We don't handle character and strings so far. Which part of the instruction set much we change in order to handle them?

SUMMARY OF 4 CATEGORIES OF MACHINE

Machine	Instruction Loading	Instruction Storage	Instruction Address & If-Goto	Data Storage, with Direct Assignment & Computation	Problem Solve
A	Manual	No	No	No	Average of 5 numbers
B	Automatic	Yes	No	No	Average of 5 numbers
C	Automatic	Yes	Yes	No	Average of N numbers with N pre-computed
D	Automatic	Yes	Yes	Yes	Average of N numbers with N computed as part of the program. Median of N numbers.

ANATOMY OF A PROGRAMMING LANGUAGE

Data Memory

Operators & Data Types

Week 2: Data Types
Week 6,8: Pointers and Array
Week 9: Characters and String
Week 12: Structure

Program Memory

Control Statement(Retrieval)

Week 5: Selection Statement (if)
Repetition Statement(while)

Organization

Week 3: Functions

Input/Output

Week 2: Overview of C
Week 13: File Processing

Algorithm & Programming Methodology:

Week 1, 2: Computational Thinking and Algorithmic Problem Solving
Week 7: Searching and Sorting
Week 11: Recursion

THREE TRACKS TO LEARN PROGRAMMING

- Track 1: Follow through the course as normal
- Track 2: Self-learning from reading and examples
- Track 3: A mixture of Track 1 and Track 2. Go to track 2 as and when comfortable

HOW DO WE QUICKLY LEARN A NEW PROGRAMMING LANGUAGE

- Step 1: Get a short document or cheat sheet that summarize various syntax of the language
- Step 2: Look through and understand a program in the new programming language by
  - Analyzing the structure of the program by looking at where are the functions, loops and if-then statement. Check the dependency between variables
  - Trace through the program with a debugger analyzing the logic and looking at the change in values of the variable
- Step 3: Goto Step 2

CHEAT SHEET AND SHORT DOCUMENT(I)

C Reference Card (ANSI)

Program Structure/Functions

type fnc(*type*$_1$,...)	function declarations
type name	external variable declarations
`main() {`	main routine
declarations	local variable declarations
statements	
`}`	
type fnc(*arg*$_1$,...) {	function definition
declarations	local variable declarations
statements	
`return` *value*`;`	
`}`	
`/* */`	comments
`main(int argc, char *argv[])`	main with args
`exit(`*arg*`)`	terminate execution

C Preprocessor

include library file	`#include <`*filename*`>`
include user file	`#include "`*filename*`"`
replacement text	`#define` *name text*
replacement macro	`#define` *name*`(`*var*`)` *text*
Example. `#define max(A,B) ((A)>(B) ? (A) : (B))`	
undefine	`#undef` *name*
quoted string in replace	`#`
concatenate args and rescan	`##`
conditional execution	`#if, #else, #elif, #endif`
is *name* defined, not defined?	`#ifdef, #ifndef`
name defined?	`defined(`*name*`)`
line continuation char	`\`

Data Types/Declarations

character (1 byte)	`char`
integer	`int`
float (single precision)	`float`
float (double precision)	`double`
short (16 bit integer)	`short`
long (32 bit integer)	`long`
positive and negative	`signed`
only positive	`unsigned`
pointer to `int`, `float`,...	`*int, *float,...`
enumeration constant	`enum`
constant (unchanging) value	`const`
declare external variable	`extern`
register variable	`register`
local to source file	`static`
no value	`void`
structure	`struct`
create name by data type	`typedef` *typename*
size of an object (type is `size_t`)	`sizeof` *object*
size of a data type (type is `size_t`)	`sizeof(`*type name*`)`

Initialization

initialize variable	*type name=value*
initialize array	*type name*`[]={`*value*$_1$,...`}`
initialize char string	`char` *name*`[]="`*string*`"`

Constants

long (suffix)	`L` or `l`
float (suffix)	`F` or `f`
exponential form	`e`
octal (prefix zero)	`0`
hexadecimal (prefix zero-ex)	`0x` or `0X`
character constant (char, octal, hex)	`'a', '\ooo', '\xhh'`
newline, cr, tab, backspace	`\n, \r, \t, \b`
special characters	`\\, \?, \', \"`
string constant (ends with `'\0'`)	`"abc...de"`

Pointers, Arrays & Structures

declare pointer to *type*	*type* `*`*name*
declare function returning pointer to *type*	*type* `*f()`
declare pointer to function returning *type*	*type* `(*pf)()`
generic pointer type	`void *`
null pointer	`NULL`
object pointed to by *pointer*	`*`*pointer*
address of object *name*	`&`*name*
array	*name*`[`*dim*`]`
multi-dim array	*name*`[`*dim*$_1$`][`*dim*$_2$`]...`

Structures

`struct` *tag* `{`	structure template
declarations	declaration of members
`};`	
create structure	`struct` *tag name*
member of structure from template	*name*`.`*member*
member of pointed to structure	*pointer* `->` *member*
Example. `(*p).x` and `p->x` are the same	
single value, multiple type structure	`union`
bit field with *b* bits	*member* `:` *b*

Operators (grouped by precedence)

structure member operator	*name*`.`*member*
structure pointer	*pointer*`->`*member*
increment, decrement	`++, --`
plus, minus, logical not, bitwise not	`+, -, !, ~`
indirection via pointer, address of object	`*`*pointer*`, &`*name*
cast expression to type	`(`*type*`)` *expr*
size of an object	`sizeof`
multiply, divide, modulus (remainder)	`*, /, %`
add, subtract	`+, -`
left, right shift [bit ops]	`<<, >>`
comparisons	`>, >=, <, <=`
comparisons	`==, !=`
bitwise and	`&`
bitwise exclusive or	`^`
bitwise or (incl)	`|`
logical and	`&&`
logical or	`||`
conditional expression	*expr*$_1$ `?` *expr*$_2$ `:` *expr*$_3$
assignment operators	`+=, -=, *=, ...`
expression evaluation separator	`,`

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

Flow of Control

statement terminator	`;`
block delimeters	`{ }`
exit from `switch`, `while`, `do`, `for`	`break`
next iteration of `while`, `do`, `for`	`continue`
go to	`goto` *label*
label	*label*`:`
return value from function	`return` *expr*

Flow Constructions

`if` statement	`if (`*expr*`)` *statement* `else if (`*expr*`)` *statement* `else` *statement*
`while` statement	`while (`*expr*`)` *statement*
`for` statement	`for (`*expr*$_1$`;` *expr*$_2$`;` *expr*$_3$`)` *statement*
`do` statement	`do` *statement* `while(`*expr*`);`

`switch` statement	`switch (`*expr*`) {`
	`case` *const*$_1$`:` *statement*$_1$ `break;`
	`case` *const*$_2$`:` *statement*$_2$ `break;`
	`default:` *statement*
	`}`

ANSI Standard Libraries

`<assert.h>`	`<ctype.h>`	`<errno.h>`	`<float.h>`	`<limits.h>`
`<locale.h>`	`<math.h>`	`<setjmp.h>`	`<signal.h>`	`<stdarg.h>`
`<stddef.h>`	`<stdio.h>`	`<stdlib.h>`	`<string.h>`	`<time.h>`

Character Class Tests <ctype.h>

alphanumeric?	`isalnum(c)`
alphabetic?	`isalpha(c)`
control character?	`iscntrl(c)`
decimal digit?	`isdigit(c)`
printing character (not incl space)?	`isgraph(c)`
lower case letter?	`islower(c)`
printing character (incl space)?	`isprint(c)`
printing char except space, letter, digit?	`ispunct(c)`
space, formfeed, newline, cr, tab, vtab?	`isspace(c)`
upper case letter?	`isupper(c)`
hexadecimal digit?	`isxdigit(c)`
convert to lower case?	`tolower(c)`
convert to upper case?	`toupper(c)`

String Operations <string.h>

`s,t` are strings, `cs,ct` are constant strings

length of `s`	`strlen(s)`
copy `ct` to `s`	`strcpy(s,ct)`
up to `n` chars	`strncpy(s,ct,n)`
concatenate `ct` after `s`	`strcat(s,ct)`
up to `n` chars	`strncat(s,ct,n)`
compare `cs` to `ct`	`strcmp(cs,ct)`
only first `n` chars	`strncmp(cs,ct,n)`
pointer to first `c` in `cs`	`strchr(cs,c)`
pointer to last `c` in `cs`	`strrchr(cs,c)`
copy `n` chars from `ct` to `s`	`memcpy(s,ct,n)`
copy `n` chars from `ct` to `s` (may overlap)	`memmove(s,ct,n)`
compare `n` chars of `cs` with `ct`	`memcmp(cs,ct,n)`
pointer to first `c` in first `n` chars of `cs`	`memchr(cs,c,n)`
put `c` into first `n` chars of `cs`	`memset(s,c,n)`

CHEAT SHEET AND SHORT DOCUMENT(II)

C Reference Card (ANSI)

Input/Output <stdio.h>

Standard I/O

standard input stream	stdin
standard output stream	stdout
standard error stream	stderr
end of file	EOF
get a character	getchar()
print a character	putchar(*chr*)
print formatted data	printf("*format*",arg_1,...)
print to string s	sprintf(s,"*format*",arg_1,...)
read formatted data	scanf("*format*",&$name_1$,...)
read from string s	sscanf(s,"*format*",&$name_1$,...)
read line to string s (< max chars)	gets(s,max)
print string s	puts(s)

File I/O

declare file pointer	FILE *fp
pointer to named file	fopen("*name*","*mode*")
modes: r (read), w (write), a (append)	
get a character	getc(*fp*)
write a character	putc(*chr*,*fp*)
write to file	fprintf(*fp*,"*format*",arg_1,...)
read from file	fscanf(*fp*,"*format*",arg_1,...)
close file	fclose(*fp*)
non-zero if error	ferror(*fp*)
non-zero if EOF	feof(*fp*)
read line to string s (< max chars)	fgets(s,max,*fp*)
write string s	fputs(s,*fp*)

Codes for Formatted I/O: "%-+ 0*w.pmc*"

-	left justify
+	print with sign
space	print space if no sign
0	pad with leading zeros
w	min field width
p	precision
m	conversion character: h short, l long, L long double
c	conversion character:
d,i	integer
u	unsigned
c	single char
s	char string
f	double
e,E	exponential
o	octal
x,X	hexadecimal
p	pointer
n	number of chars written
g,G	same as f or e,E depending on exponent

Variable Argument Lists <stdarg.h>

declaration of pointer to arguments	va_list *name*;
initialization of argument pointer	va_start(*name*,*lastarg*)
lastarg is last named parameter of the function	
access next unamed arg, update pointer	va_arg(*name*,*type*)
call before exiting function	va_end(*name*)

Standard Utility Functions <stdlib.h>

absolute value of int n	abs(n)
absolute value of long n	labs(n)
quotient and remainder of ints n,d	div(n,d)
retursn structure with div_t.quot and div_t.rem	
quotient and remainder of longs n,d	ldiv(n,d)
returns structure with ldiv_t.quot and ldiv_t.rem	
pseudo-random integer [0,RAND_MAX]	rand()
set random seed to n	srand(n)
terminate program execution	exit(status)
pass string s to system for execution	system(s)

Conversions

convert string s to double	atof(s)
convert string s to integer	atoi(s)
convert string s to long	atol(s)
convert prefix of s to double	strtod(s,endp)
convert prefix of s (base b) to long	strtol(s,endp,b)
same, but unsigned long	strtoul(s,endp,b)

Storage Allocation

allocate storage	malloc(size), calloc(nobj,size)
change size of object	realloc(pts,size)
deallocate space	free(ptr)

Array Functions

search array for key	bsearch(key,array,n,size,cmp())
sort array ascending order	qsort(array,n,size,cmp())

Time and Date Functions <time.h>

processor time used by program	clock()
Example. clock()/CLOCKS_PER_SEC is time in seconds	
current calendar time	time()
$time_2$-$time_1$ in seconds (double)	difftime($time_2$,$time_1$)
arithmetic types representing times	clock_t,time_t
structure type for calendar time comps	tm

tm_sec	seconds after minute
tm_min	minutes after hour
tm_hour	hours since midnight
tm_mday	day of month
tm_mon	months since January
tm_year	years since 1900
tm_wday	days since Sunday
tm_yday	days since January 1
tm_isdst	Daylight Savings Time flag

convert local time to calendar time	mktime(tp)
convert time in tp to string	asctime(tp)
convert calendar time in tp to local time	ctime(tp)
convert calendar time to GMT	gmtime(tp)
convert calendar time to local time	localtime(tp)
format date and time info	strftime(s,smax,"*format*",tp)
tp is a pointer to a structure of type tm	

Mathematical Functions <math.h>

Arguments and returned values are double

trig functions	sin(x), cos(x), tan(x)
inverse trig functions	asin(x), acos(x), atan(x)
arctan(y/x)	atan2(y,x)
hyperbolic trig functions	sinh(x), cosh(x), tanh(x)
exponentials & logs	exp(x), log(x), log10(x)
exponentials & logs (2 power)	ldexp(x,n), frexp(x,*e)
division & remainder	modf(x,*ip), fmod(x,y)
powers	pow(x,y), sqrt(x)
rounding	ceil(x), floor(x), fabs(x)

Integer Type Limits <limits.h>

The numbers given in parentheses are typical values for the constants on a 32-bit Unix system.

CHAR_BIT	bits in char	(8)
CHAR_MAX	max value of char	(127 or 255)
CHAR_MIN	min value of char	(−128 or 0)
INT_MAX	max value of int	(+32,767)
INT_MIN	min value of int	(−32,768)
LONG_MAX	max value of long	(+2,147,483,647)
LONG_MIN	min value of long	(−2,147,483,648)
SCHAR_MAX	max value of signed char	(+127)
SCHAR_MIN	min value of signed char	(−128)
SHRT_MAX	max value of short	(+32,767)
SHRT_MIN	min value of short	(−32,768)
UCHAR_MAX	max value of unsigned char	(255)
UINT_MAX	max value of unsigned int	(65,535)
ULONG_MAX	max value of unsigned long	(4,294,967,295)
USHRT_MAX	max value of unsigned short	(65,536)

Float Type Limits <float.h>

FLT_RADIX	radix of exponent rep	(2)
FLT_ROUNDS	floating point rounding mode	
FLT_DIG	decimal digits of precision	(6)
FLT_EPSILON	smallest x so $1.0 + x \neq 1.0$	(10^{-5})
FLT_MANT_DIG	number of digits in mantissa	
FLT_MAX	maximum floating point number	(10^{37})
FLT_MAX_EXP	maximum exponent	
FLT_MIN	minimum floating point number	(10^{-37})
FLT_MIN_EXP	minimum exponent	
DBL_DIG	decimal digits of precision	(10)
DBL_EPSILON	smallest x so $1.0 + x \neq 1.0$	(10^{-9})
DBL_MANT_DIG	number of digits in mantissa	
DBL_MAX	max double floating point number	(10^{37})
DBL_MAX_EXP	maximum exponent	
DBL_MIN	min double floating point number	(10^{-37})
DBL_MIN_EXP	minimum exponent	

Send comments and corrections to J.H. Silverman, Math. Dept., Brown Univ., Providence, RI 02912 USA. (jhs@math.brown.edu)

CHEAT SHEET AND SHORT DOCUMENT(III)

Essential C

By Nick Parlante

This Stanford CS Education document tries to summarize all the basic features of the C language. The coverage is pretty quick, so it is most appropriate as review or for someone with some programming background in another language. Topics include variables, int types, floating point types, promotion, truncation, operators, control structures (if, while, for), functions, value parameters, reference parameters, structs, pointers, arrays, the pre-processor, and the standard C library functions.

The most recent version is always maintained at its Stanford CS Education Library URL `http://cslibrary.stanford.edu/101/`. Please send your comments to nick.parlante@cs.stanford.edu.

I hope you can share and enjoy this document in the spirit of goodwill in which it is given away -- Nick Parlante, 4/2003, Stanford California.

Table of Contents

The C Language

C is a professional programmer's language. It was designed to get in one's way as little as possible. Kernighan and Ritchie wrote the original language definition in their book, *The C Programming Language* (below), as part of their research at AT&T. Unix and C++ emerged from the same labs. For several years I used AT&T as my long distance carrier in appreciation of all that CS research, but hearing "thank you for using AT&T" for the millionth time has used up that good will.

Section 1
Basic Types and Operators

C provides a standard, minimal set of basic data types. Sometimes these are called "primitive" types. More complex data structures can be built up from these basic types.

Integer Types

The "integral" types in C form a family of integer types. They all behave like integers and can be mixed together and used in similar ways. The differences are due to the different number of bits ("widths") used to implement each type -- the wider types can store a greater ranges of values.

`char` ASCII character -- at least 8 bits. Pronounced "car". As a practical matter `char` is basically always a byte which is 8 bits which is enough to store a single ASCII character. 8 bits provides a signed range of -128..127 or an unsigned range is 0..255. `char` is also required to be the "smallest addressable unit" for the machine -- each byte in memory has its own address.

`short` Small integer -- at least 16 bits which provides a signed range of -32768..32767. Typical size is 16 bits. Not used so much.

`int` Default integer -- at least 16 bits, with 32 bits being typical. Defined to be the "most comfortable" size for the computer. If you do not really care about the range for an integer variable, declare it `int` since that is likely to be an appropriate size (16 or 32 bit) which works well for that machine.

`long` Large integer -- at least 32 bits. Typical size is 32 bits which gives a signed range of about -2 billion ..+2 billion. Some compilers support "long long" for 64 bit ints.

The integer types can be preceded by the qualifier `unsigned` which disallows representing negative numbers, but doubles the largest positive number representable. For example, a 16 bit implementation of `short` can store numbers in the range -32768..32767, while `unsigned short` can store 0..65535. You can think of pointers as being a form of `unsigned long` on a machine with 4 byte pointers. In my opinion, it's best to avoid using `unsigned` unless you really need to. It tends to cause more misunderstandings and problems than it is worth.

Extra: Portability Problems

Instead of defining the exact sizes of the integer types, C defines lower bounds. This makes it easier to implement C compilers on a wide range of hardware. Unfortunately it occasionally leads to bugs where a program runs differently on a 16-bit-int machine than it runs on a 32-bit-int machine. In particular, if you are designing a function that will be implemented on several different machines, it is a good idea to use typedefs to set up types like `Int32` for 32 bit int and `Int16` for 16 bit int. That way you can prototype a function `Foo(Int32)` and be confident that the typedefs for each machine will be set so that the function really takes exactly a 32 bit int. That way the code will behave the same on all the different machines.

char Constants

A `char` constant is written with single quotes (') like 'A' or 'z'. The `char` constant 'A' is really just a synonym for the ordinary integer value 65 which is the ASCII value for

CHEAT SHEET AND SHORT DOCUMENT(IV)

Python 3 Beginner's Reference Cheat Sheet

Alvaro Sebastian
http://www.sixthresearcher.com

Main data types

boolean = *True* / *False*
integer = 10
float = 10.01
string = "123abc"
list = [value1, value2, ...]
dictionary = { key1:value1, key2:value2, ...}

Numeric operators

+	addition
-	subtraction
*	multiplication
/	division
**	exponent
%	modulus
//	floor division

Comparison operators

==	equal
!=	different
>	higher
<	lower
>=	higher or equal
<=	lower or equal

Boolean operators

and	logical AND
or	logical OR
not	logical NOT

Special characters

#	coment
\n	new line
\<char>	scape char

String operations

string[i]	retrieves character at position i
string[-1]	retrieves last character
string[i:j]	retrieves characters in range i to j

List operations

list = []	defines an empty list
list[i] = x	stores x with index i
list[i]	retrieves the item with index I
list[-1]	retrieves last item
list[i:j]	retrieves items in the range i to j
del list[i]	removes the item with index i

Dictionary operations

dict = {}	defines an empty dictionary
dict[k] = x	stores x associated to key k
dict[k]	retrieves the item with key k
del dict[k]	removes the item with key k

String methods

string.upper()	converts to uppercase
string.lower()	converts to lowercase
string.count(x)	counts how many times x appears
string.find(x)	position of the x first occurrence
string.replace(x,y)	replaces x for y
string.strip(x)	returns a list of values delimited by x
string.join(L)	returns a string with L values joined by string
string.format(x)	returns a string that includes formatted x

List methods

list.append(x)	adds x to the end of the list
list.extend(L)	appends L to the end of the list
list.insert(i,x)	inserts x at i position
list.remove(x)	removes the first list item whose value is x
list.pop(i)	removes the item at position i and returns its value
list.clear()	removes all items from the list
list.index(x)	returns a list of values delimited by x
list.count(x)	returns a string with list values joined by S
list.sort()	sorts list items
list.reverse()	reverses list elements
list.copy()	returns a copy of the list

Dictionary methods

dict.keys()	returns a list of keys
dict.values()	returns a list of values
dict.items()	returns a list of pairs (key,value)
dict.get(k)	returns the value associtated to the key k
dict.pop()	removes the item associated to the key and returns its value
dict.update(D)	adds keys-values (D) to dictionary
dict.clear()	removes all keys-values from the dictionary
dict.copy()	returns a copy of the dictionary

Legend: x,y stand for any kind of data values, **s** for a string, **n** for a number, **L** for a list where **i,j** are list indexes, **D** stands for a dictionary and **k** is a dictionary key.

CHEAT SHEET AND SHORT DOCUMENT(V)

Python 3 Beginner's Reference Cheat Sheet

Alvaro Sebastian
http://www.sixthresearcher.com

Built-in functions

print(x, sep='y')	prints x objects separated by y
input(s)	prints s and waits for an input that will be returned
len(x)	returns the length of x (s, L or D)
min(L)	returns the minimum value in L
max(L)	returns the maximum value in L
sum(L)	returns the sum of the values in L
range(n1,n2,n)	returns a sequence of numbers from n1 to n2 in steps of n
abs(n)	returns the absolute value of n
round(n1,n)	returns the n1 number rounded to n digits
type(x)	returns the type of x (string, float, list, dict ...)
str(x)	converts x to string
list(x)	converts x to a list
int(x)	converts x to a integer number
float(x)	converts x to a float number
help(s)	prints help about x
map(function, L)	Applies function to values in L

Conditional statements

```
if <condition>:
    <code>
else if <condition>:
    <code>
...
else:
    <code>

if <value> in <list>:
```

Data validation

```
try:
    <code>
except <error>:
    <code>
else:
    <code>
```

Working with files and folders

```
import os
os.getcwd()
os.makedirs(<path>)
os.chdir(<path>)
os.listdir(<path>)
```

Loops

```
while <condition>:
    <code>

for <variable> in <list>:
    <code>

for <variable> in
range(start,stop,step):
    <code>

for key, value in
dict.items():
    <code>
```

Loop control statements

break	finishes loop execution
continue	jumps to next iteration
pass	does nothing

Running external programs

```
import os
os.system(<command>)
```

Functions

```
def function(<params>):
    <code>
    return <data>
```

Modules

```
import module
module.function()

from module import *
function()
```

Reading and writing files

```
f = open(<path>,'r')
f.read(<size>)
f.readline(<size>)
f.close()

f = open(<path>,'r')
for line in f:
    <code>
f.close()

f = open(<path>,'w')
f.write(<str>)
f.close()
```

Legend: x,y stand for any kind of data values, **s** for a string, **n** for a number, **L** for a list where **i,j** are list indexes, **D** stands for a dictionary and **k** is a dictionary key.

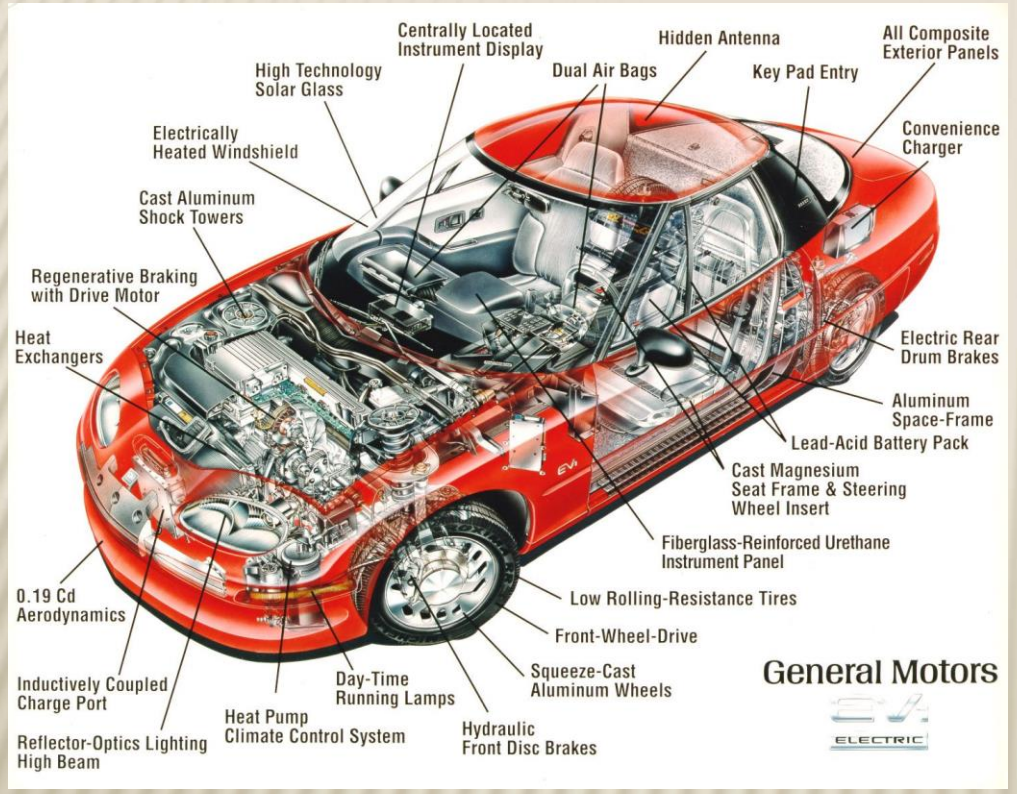
USE OF THE CHEAT SHEET

- Please keep the cheat sheet as a “roadmap” and mark those content that will be taught to you through this course
- Not everything in the cheat sheet will be covered in this course, but at least you know what you don’t know!

HOW DO WE QUICKLY LEARN A NEW PROGRAMMING LANGUAGE

- Step 1: Get a short document or cheat sheet that summarize various syntax of the language
- Step 2: Look through and understand a program in the new programming language by
  - Analyzing the structure of the program by looking at where are the functions, loops and if-then statement. Check the dependency between variables
  - Trace through the program with a debugger analyzing the logic and looking at the change in values of the variable
- Step 3: Goto Step 2

DON'T WORRY IF YOU DON'T COMPLETELY UNDERSTAND NOW!



- We are now just looking at an overview of the car before going into details of each part of the car later
- But seeing the whole car now give you a better helicopter view of what we are learning
- We will come back to Bubblesort at the later part of the lesson!

EXAMPLE: BUBBLE SORT

```
#include <stdio.h>
#define SIZE 10
void bubbleSort(int * const array, const int size);
void swap(int *element1Ptr, int *element2Ptr);
int main(void){

  int arr[SIZE] = {2, 3, 9, 4, 1, 5, 7, 12, 18, 15};

  printf("Original array values\n\n");
  for(int i=0; i<SIZE; i++){
    printf("%d\n", arr[i]);
  }
   bubbleSort(arr, SIZE);
   printf("\n\nSorted array values\n\n");
  for(int i=0; i<SIZE; i++){
    printf("%d\n", arr[i]);
  }
 }
```

```
void bubbleSort(int * const array, const int size){
  int pass, i=0;
  for(pass=0; pass < size-1; pass++){
     for(i=0; i < size-1; i++){
       if(array[i] > array[i+1]){
        swap(&array[i], &array[i+1]);
       }
     }
  }
}

void swap(int *element1Ptr, int *element2Ptr){
  int hold;
  hold = *element1Ptr;
  *element1Ptr = *element2Ptr;
  *element2Ptr = hold;
}
```

EXAMPLE: BUBBLE SORT (OVERALL FUNCTION STRUCTURE)

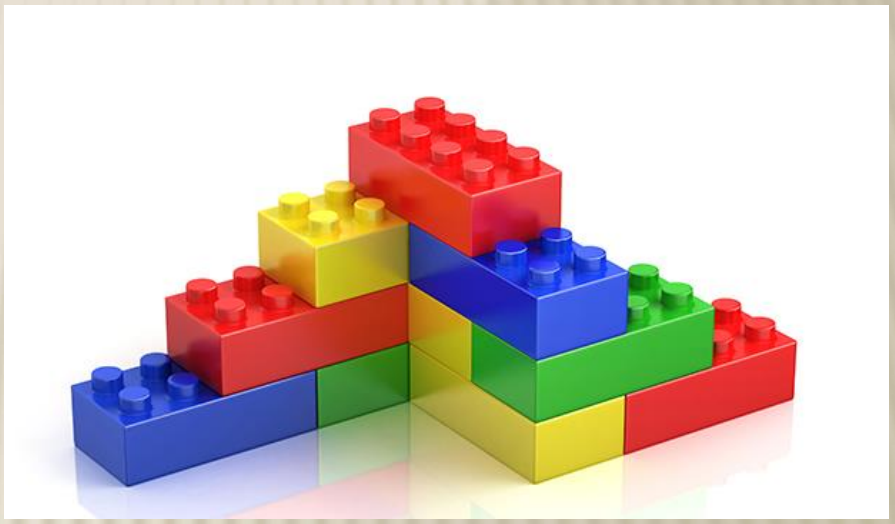
main()

↓

bubbleSort()

↓

swap()



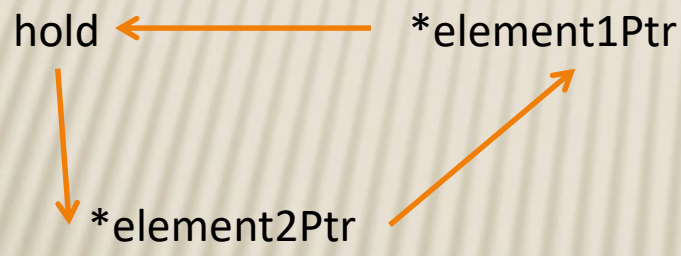
EXAMPLE: BUBBLE SORT(VARIABLE)

```
#include <stdio.h>
#define SIZE 10
void bubbleSort(int * const array, const int size);
void swap(int *element1Ptr, int *element2Ptr);
int main(void){

 int arr[SIZE] = {2, 3, 9, 4, 1, 5, 7, 12, 18, 15};

  printf("Original array values\n\n");
  for(int i= 0; i<SIZE; i++){
    printf("%d\n", arr[i]);
  }
   bubbleSort(arr, SIZE);
   printf("\n\nSorted array values\n\n");
  for(int i=0; i<SIZE; i++){
    printf("%d\n", arr[i]);
  }
 }
```

```
void bubbleSort(int * const array, const int size){
  int pass, i=0;
  for(pass=0; pass < size-1; pass++){
     for(i=0; i < size-1; i++){
       if(array[i] > array[i+1]){
        swap(&array[i], &array[i+1]);
      }
    }
  }
}

void swap(int *element1Ptr, int *element2Ptr){
  int hold;
  hold = *element1Ptr;
  *element1Ptr = *element2Ptr;
  *element2Ptr = hold;
}
```

EXAMPLE: BUBBLE SORT(VARIABLE DEPENDENCY)

```
void swap(int *element1Ptr, int *element2Ptr){
   int hold;
   hold = *element1Ptr;
   *element1Ptr = *element2Ptr;
   *element2Ptr = hold;
}
```



We draw an arrow from variable A to variable B if the value of A was transferred to B

EXAMPLE: BUBBLE SORT(CONTROL STRUCTURE)

```
void bubbleSort(int * const array, const int size){
  int pass, i=0;
  for(pass=0; pass < size-1; pass++){
    for(i=0; i < size-1; i++){
      if(array[i] > array[i+1]){
        swap(&array[i], &array[i+1]);
      }
    }
  }
}
```

Loop size-1 times, increasing pass by one every time, so pass=0,…, size-1

Do swap only if array[i]>array[i+1]

How many time will this statement be executed?

Loop size-1 times, increasing i by one every time, so i=0,…, size-1

(size-1)*(size-1)!

	i=0	i=1	…….	i=size-1
pass=0				
pass=1				
pass=2				
…….				
pass=size-1				

EXAMPLE: BUBBLE SORT(CONTROL STRUCTURE)

```
void bubbleSort(int * const array, const int size){
  int pass, i=0;
  for(pass=0; pass < size-1; pass++){
    for(i=0; i < size-1; i++){
      if(array[i] > array[i+1]){
        swap(&array[i], &array[i+1]);
      }
    }
  }
}
```

Loop size-1 times, increasing pass by one every time, so pass=0,…, size-1

Do swap only if array[i]>array[i+1]

Loop size-1 times, increasing i by one every time, so i=0,…, size-1

i=0 i=1 i=2 i=9 (i.e. size-1)

array : {2, 3, 9, 4, 1, 5, 7, 12, 18, 15} pass=0

array : {2, 3, 4, 1, 5, 7, 9, 12, 15, 18} pass=1

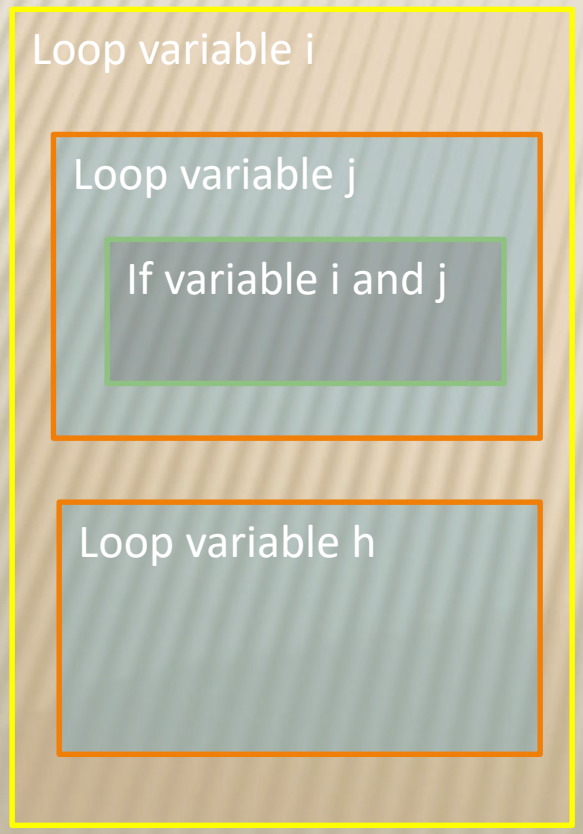
array : {2, 3, 1, 4, 5, 7, 9, 12, 15, 18} pass=2

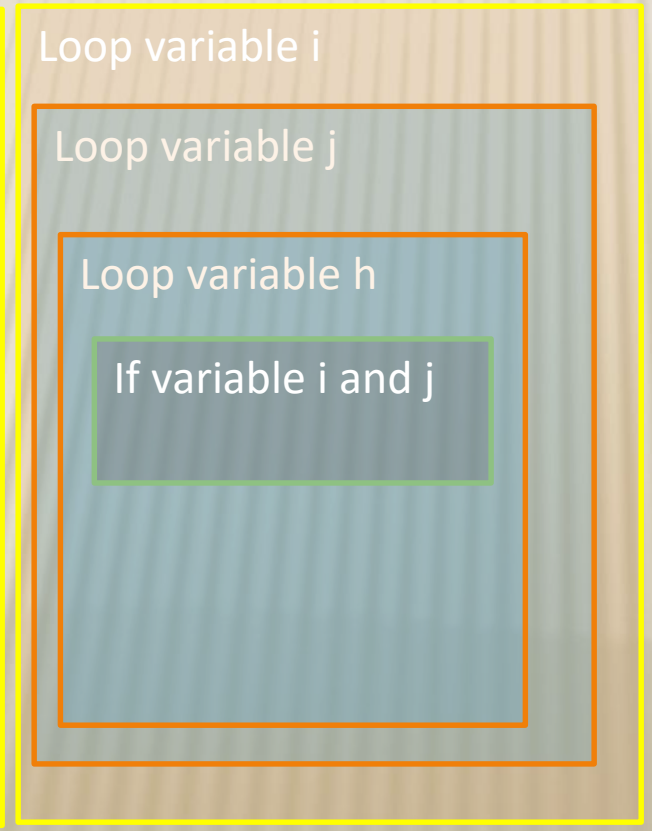
array : {2, 1, 3, 4, 5, 7, 9, 12, 15, 18} pass=3

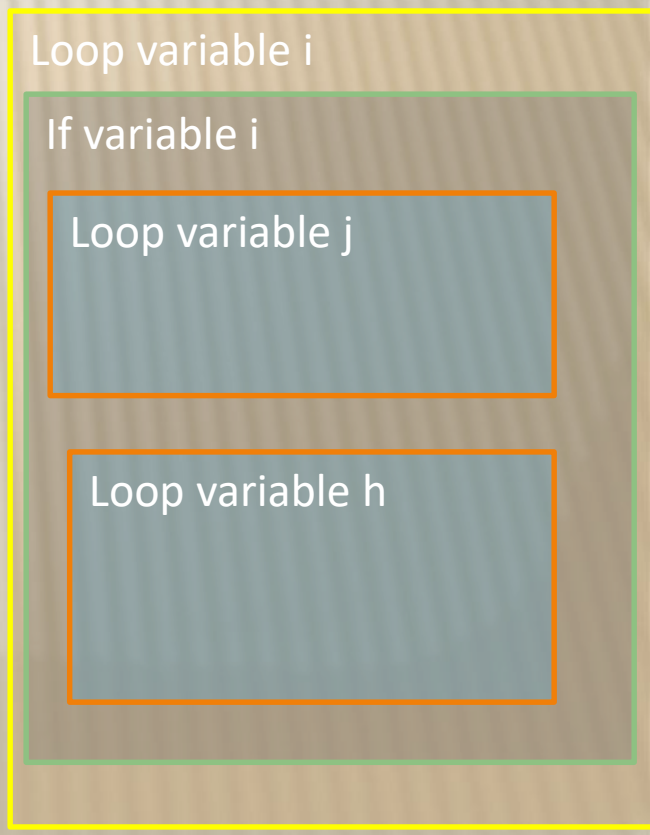
The right most passth elements in this box is always sorted by a certain pass

EXAMPLE: BUBBLE SORT(CONTROL STRUCTURE)

- You read and understand program by looking at how the control structures are organized
- Likewise, a lot of times, you design your program by deciding the organization of the control structures first!







RECAP

- We try to understand a program by
  - Figuring out the functional structures
  - Looking at what are the variables and how they are instantiated
  - Looking at variables dependency or how they affect each other
  - Looking at the control structures and check how the control variables changes within the control structure framework
- There is no hard and fast rules on which one should be done first and there is no guarantee that you know 100% what the program is doing. They are just my rule of thumbs for quickly analyzing a program

HOW DO WE QUICKLY LEARN A NEW PROGRAMMING LANGUAGE

- Step 1: Get a short document or cheat sheet that summarize various syntax of the language
- Step 2: Look through and understand a program in the new programming language by
  - Analyzing the structure of the program by look at where are the functions, loops and if-then statement. Check the dependency between variables
  - Trace through the program with a debugger analyzing the logic and looking at the change in values of the variable
- Step 3: Goto Step 2

LOGGING INTO SUNFIRE (1/2)

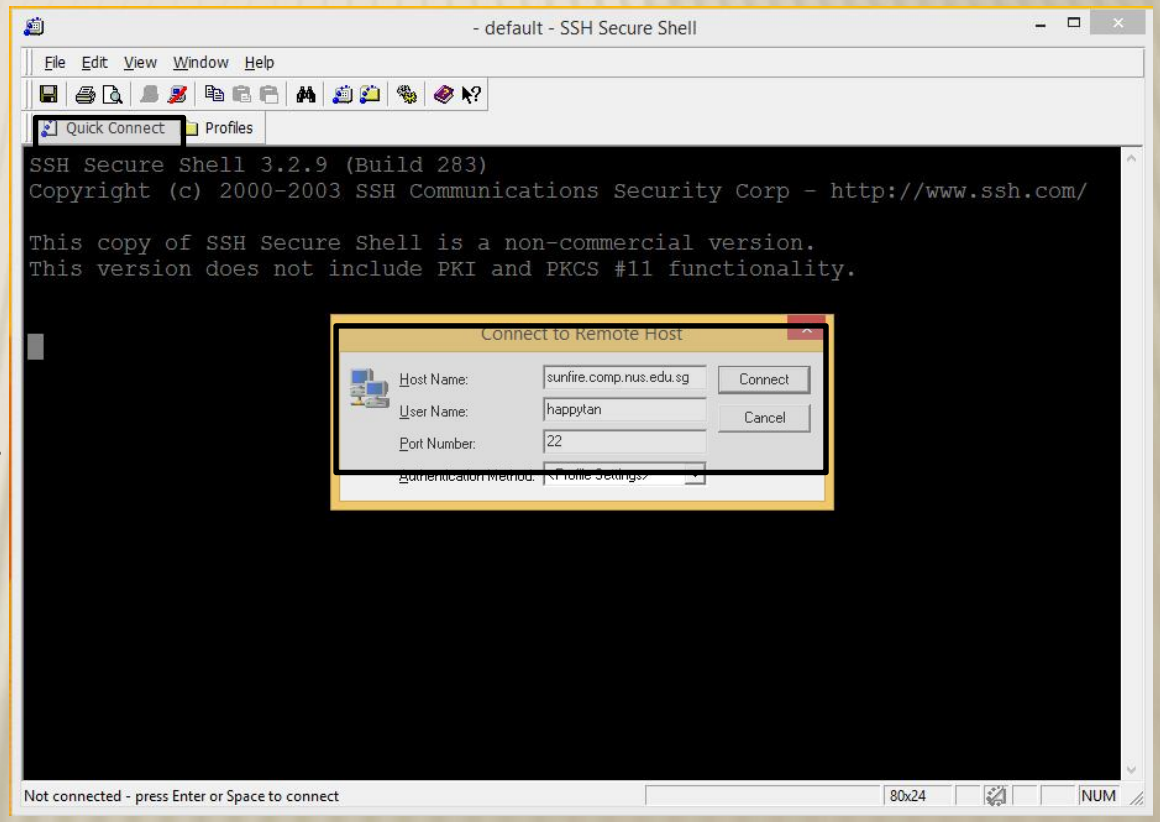
1. Look for the SSH Secure Shell Client icon or Xshell icon on your desktop, and double click on it. We shall assume you are using the former here.

or

2. Click on "Quick Connect" to get the pop-up window.

   Enter "sunfire" for Host Name if connecting within campus or "sunfire.comp.nus.edu.sg" if connecting from off campus

   Enter your UNIX id as User Name.



LOGGING INTO SUNFIRE (1/2)

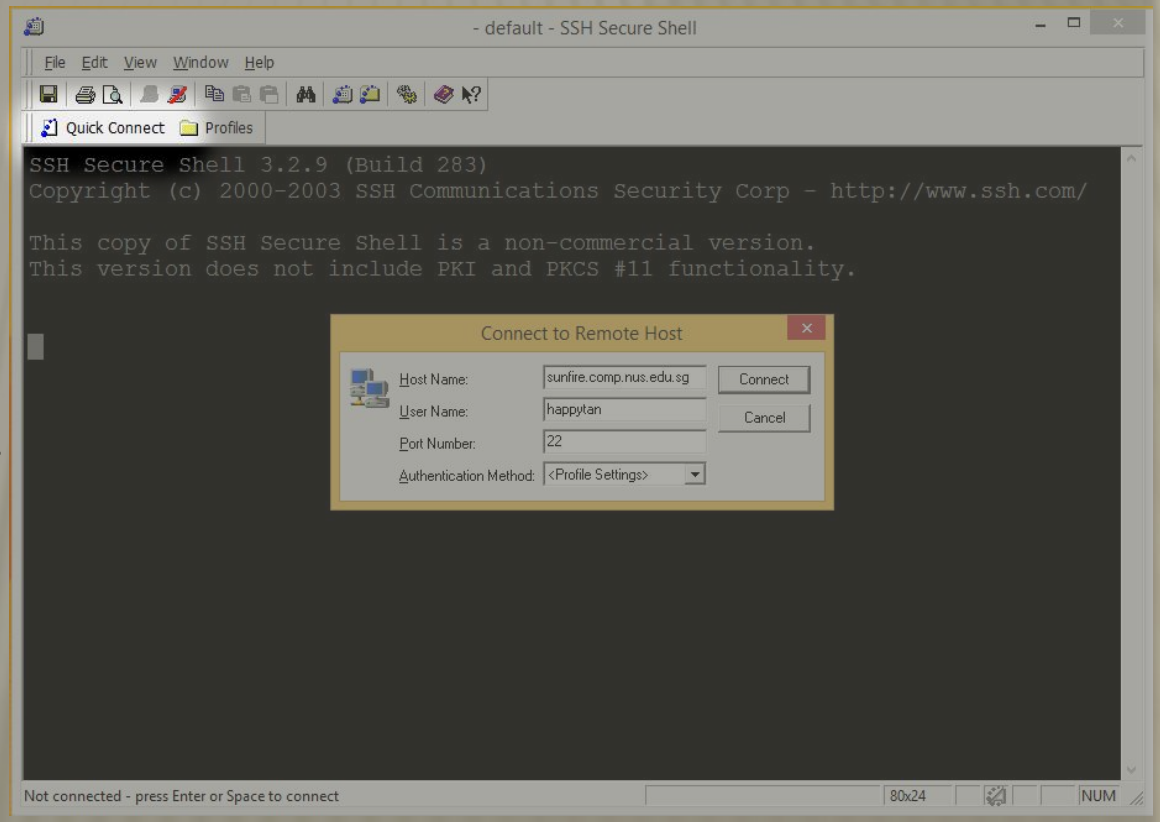
1. Look for the SSH Secure Shell Client icon or Xshell icon on your desktop, and double click on it. We shall assume you are using the former here.

or

2. Click on "Quick Connect" to get the pop-up window.

   Enter "sunfire" for Host Name if connecting within campus or "sunfire.comp.nus.edu.sg" if connecting from off campus

   Enter your UNIX id as User Name.



slide added by PowerPoint Labs

LOGGING INTO SUNFIRE (1/2)

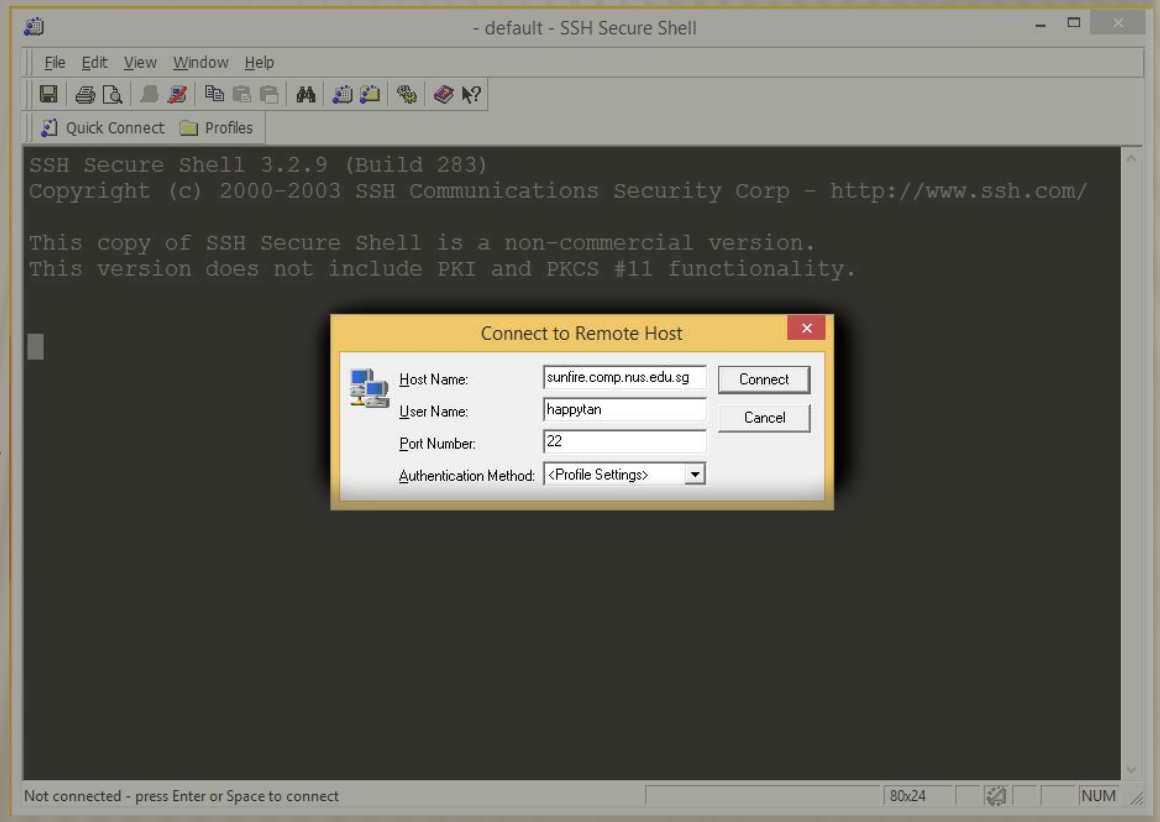
1. Look for the SSH Secure Shell Client icon or Xshell icon on your desktop, and double click on it. We shall assume you are using the former here.

or

2. Click on "Quick Connect" to get the pop-up window.

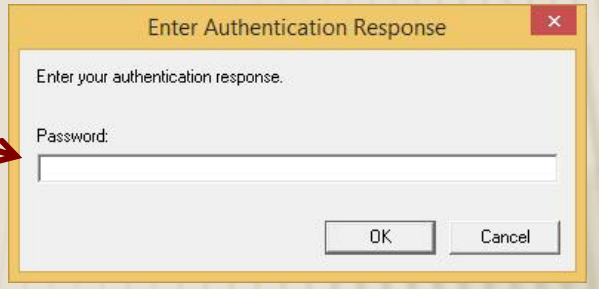
   Enter "sunfire" for Host Name if connecting within campus or "sunfire.comp.nus.edu.sg" if connecting from off campus

   Enter your UNIX id as User Name.



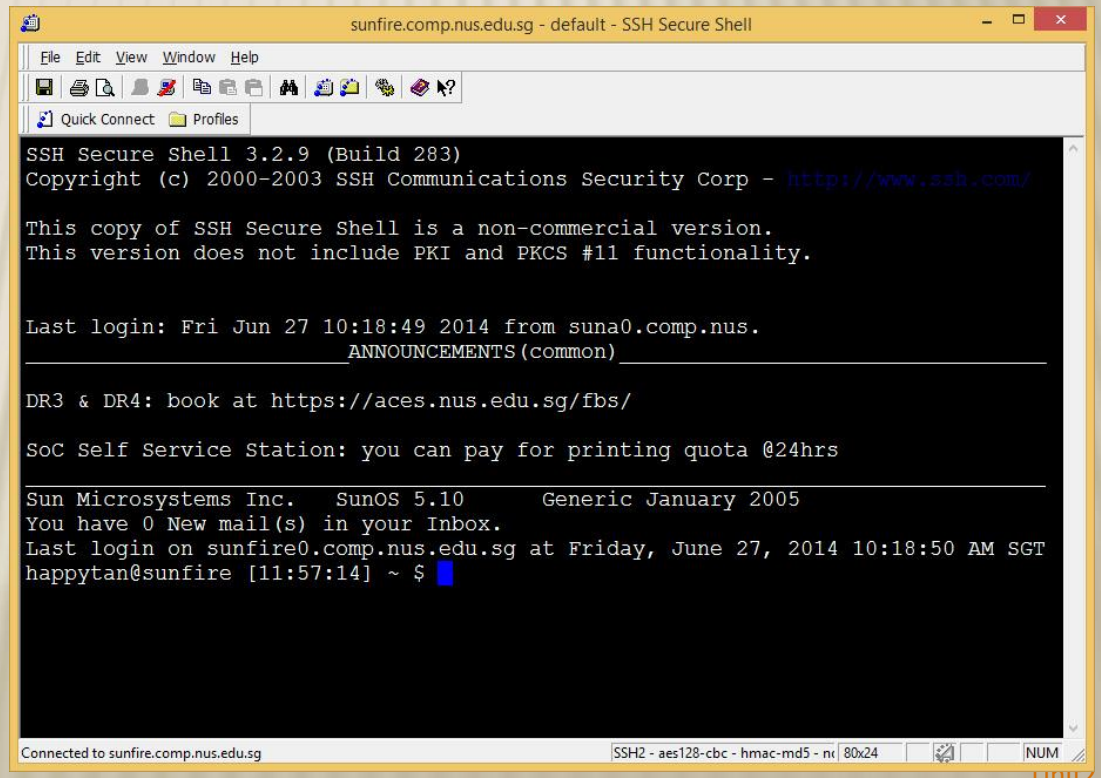
LOGGING INTO SUNFIRE (2/2)

3. Enter your UNIX password.



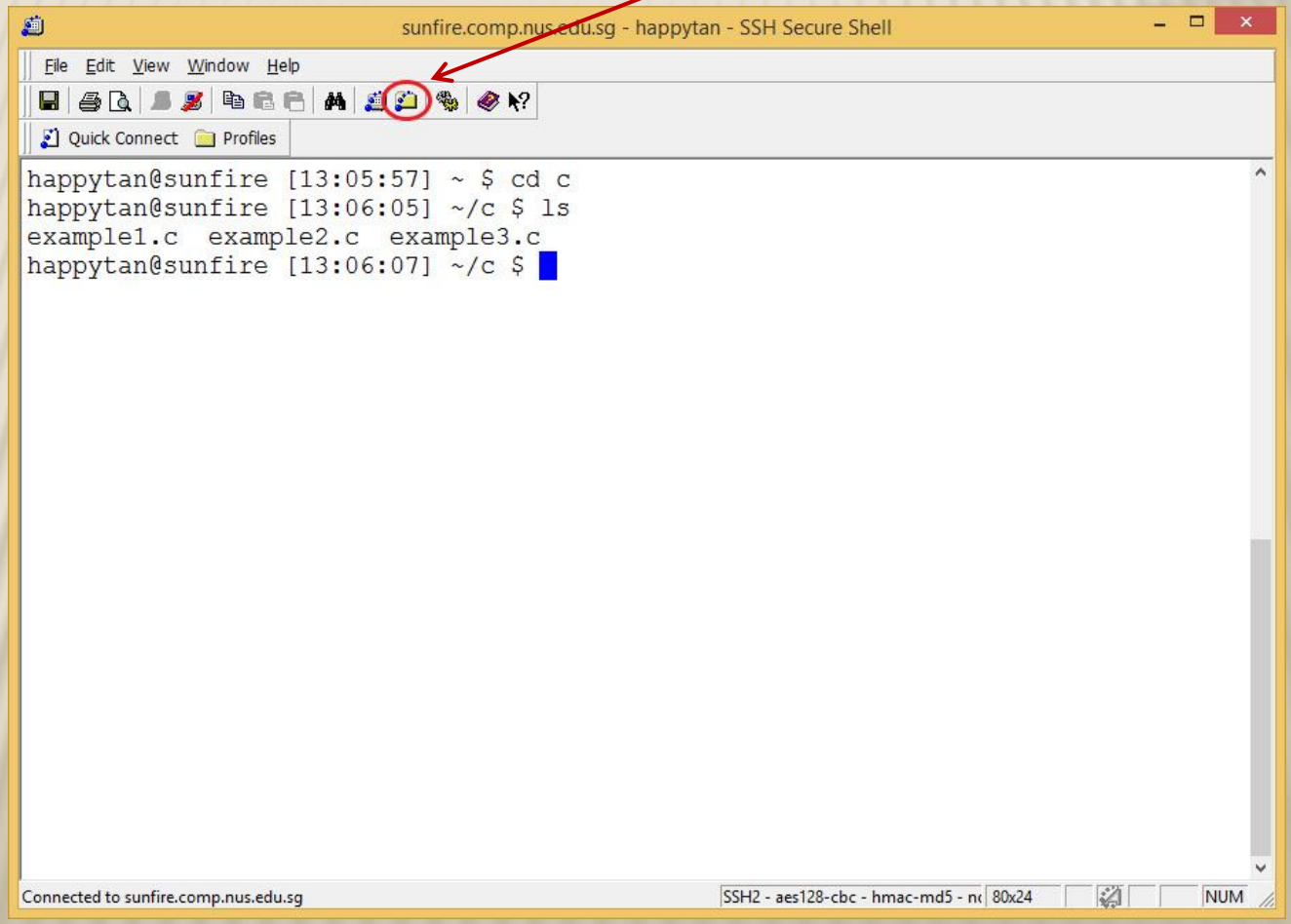
4. Once you log in successfully, you will see this screen (actual display may vary).

5. To log out from your UNIX account, type "exit" or "logout".



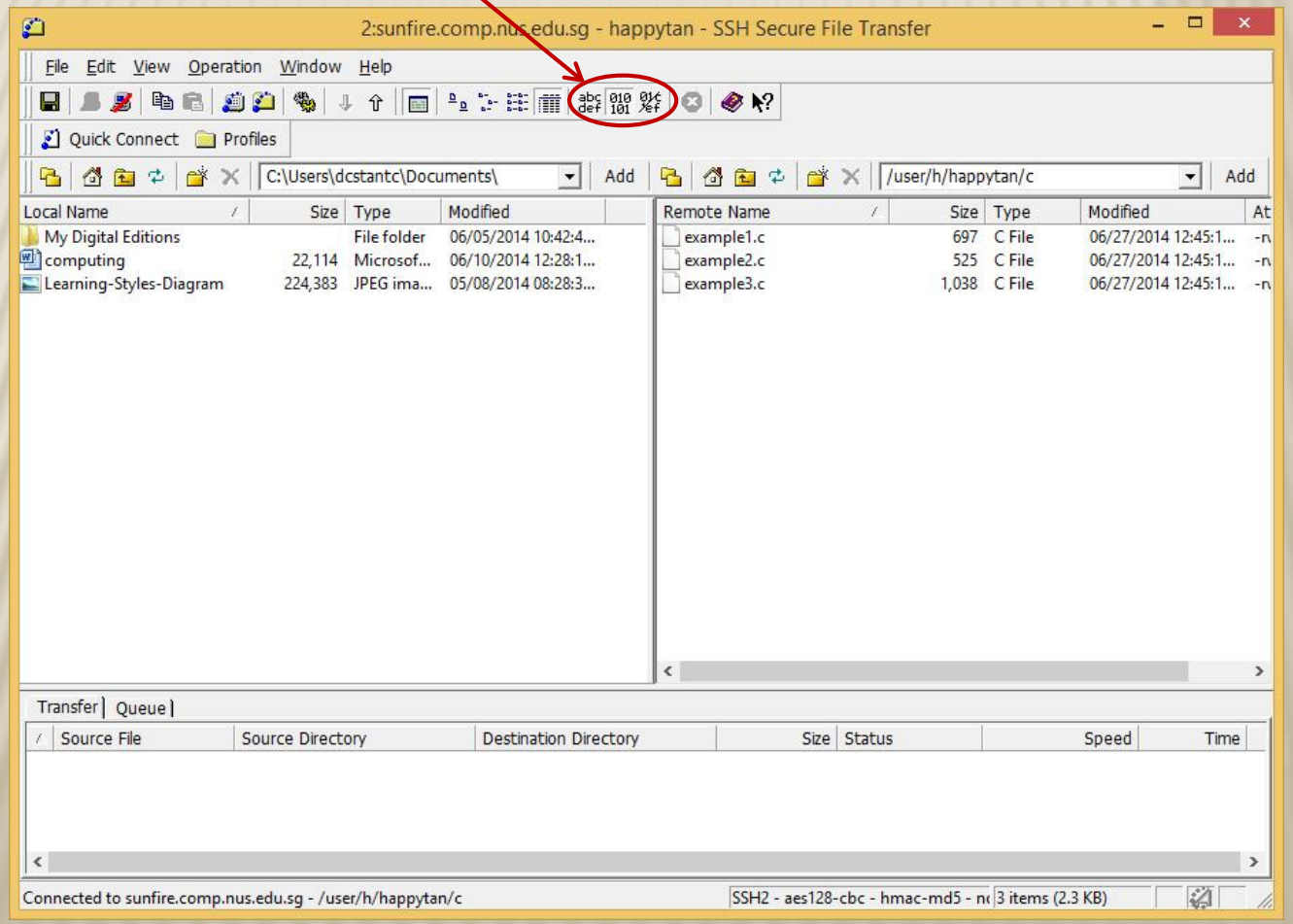
FILE TRANSFER (1/2)

- To transfer files between your sunfire account and your local computer, click on the SSH Secure File Transfer icon



FILE TRANSFER (2/2)

- Left: your local machine; right: sunfire
- Choose the format: ASCII, Binary or Auto
- Click on file(s) to transfer, and drag to the destination



USING THE GDB DEBUGGER (1/3)

- A debugger is a tool that assists in the detection and correction of errors in programs
- It usually provides the following
  - Stepping
  - Breakpoint
  - Watches (inspecting values of variables)
- We will illustrate these with gnu debugger gdb

USING THE GDB DEBUGGER (2/3)

- Step 1: Add the –g option when compiling and linking your program:
  - **`gcc -g bubblesort.c`**
- Step 2: Start gdb with your program:
  - **`gdb a.out`**
- Step 3: Use gdb commands to step through your program:
  - Break at a specific function: *break function_name*
    - Start with: *break main*
  - Run program: *run* (or *r*)
  - Step to the next line of code: *step* (or *s*)
  - List source code: *list* (or *l*)
  - Examine the value of a variable: *print variable_name*
  - Quit the debugger: *quit*

Explore other gdb commands on your own!

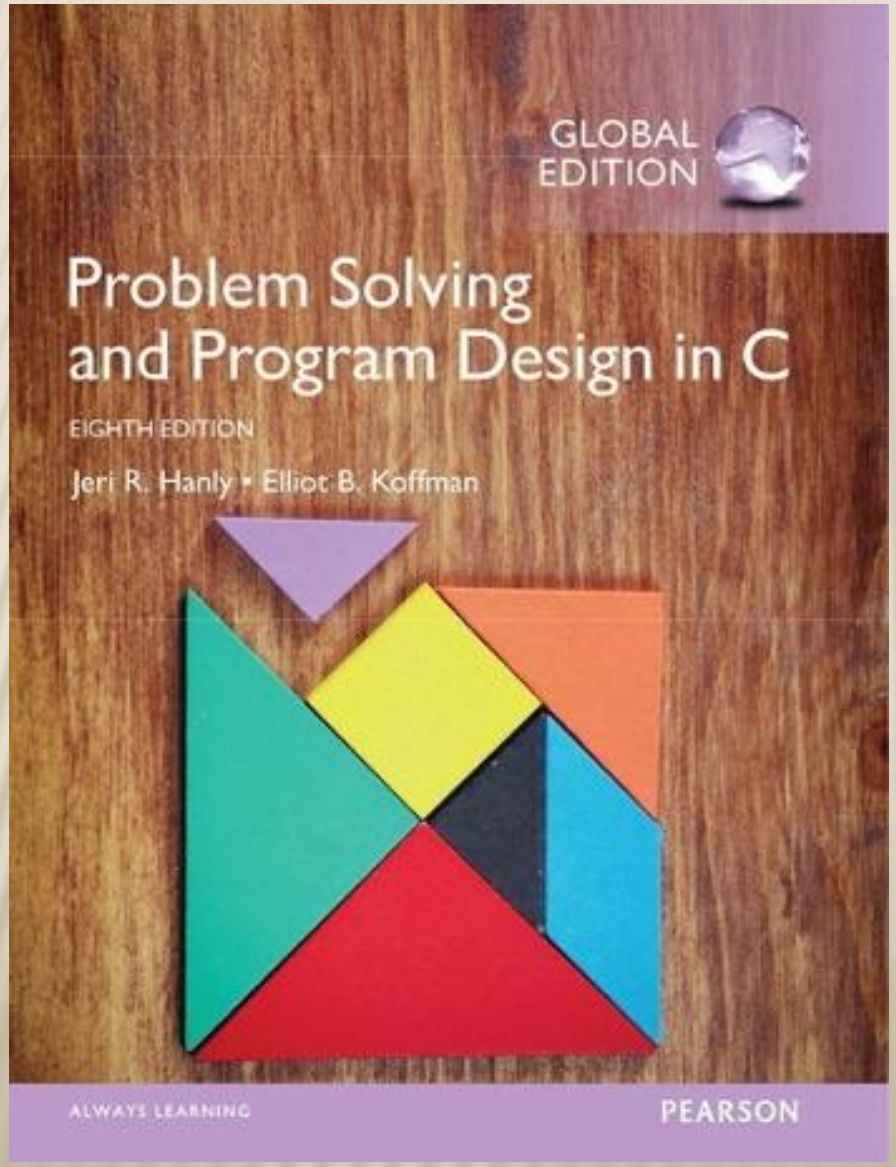
USING THE GDB DEBUGGER (3/3)

- Some links on gdb:
  - https://sourceware.org/gdb/current/onlinedocs/gdb/
  - http://www.cprogramming.com/gdb.html
  - http://www.tutorialspoint.com/gnu_debugger/

WHAT MADE A GOOD PROGRAMMER?

- Ability to compare and contrast different topics
- Ability to generalize and form frameworks
- Ability to connect problems of the same nature
- Ability to have a broad overview while managing the details
- Abilities to be a good programmer also happen to be abilities that help you to learn better and useful for your future career

TEXTBOOK


GLOBAL
EDITION
Problem Solving
and Program Design in C
EIGHTH EDITION
Jeri R. Hanly • Elliot B. Koffman
ALWAYS LEARNING
PEARSON

SCHEDULE

- Refer to Lesson Plan in IVLE
  https://luminus.nus.edu.sg/modules/788d94d3-fac3-4b11-b596-bf1a3e219f1a/lessons/semester-view

Week	Date	Topic	Labs	Tutorial	Comment
1	12 Aug 2019 - Fri, 16 Aug 2019	Introduction and Computational Thinking			
2	19 Aug 2019 - Fri, 23 Aug 2019	Algorithmic Problem Solving		X	
3	26 Aug 2019 - Fri, 30 Aug 2019	Overview of C Programming	L0 Start, L1 Start	X	
4	2 Sep 2019 - Fri, 6 Sep 2019	Functions, Program Designs			
5	9 Sep 2019 - Fri, 13 Sep 2019	Selection and Repetition Statements, Testing and Debuggin	L1 End, L2 Start	X	
6	16 Sep 2019 - Fri, 20 Sep 2019	Pointers and Array, Random Number,		X	
Recess	21 Sep 2019 - Sun, 29 Sep 201	Recess	L2 End, L3 Start		
7	30 Sep 2019 - Fri, 4 Oct 2019	Searching and Sorting		X	
8	7 Oct 2019 - Fri, 11 Oct 2019	Functions with Pointer Parameter, Multidimensional Array	L3 End, L4 Start	X	PE1
9	14 Oct 2019 - Fri, 18 Oct 2019	Characters and Strings		X	
10	21 Oct 2019 - Fri, 25 Oct 2019	Mid-Term Quiz	L4 End, L5 Start	X	Quiz
11	28 Oct 2019 - Fri, 1 Nov 2019	Recursion		X	
12	4 Nov 2019 - Fri, 8 Nov 2019	Structures and UNIX I/O Redirection	L5 End, L6 Start	X	PE2
13	11 Nov 2019 - Fri, 15 Nov 2019	File Processing and Review	L6 End	X	

ASSESSMENTS

- CA (60%)
  - Take-home lab assignments (10%)
  - Practical Exams (30%)
    - PE1(10%) **(12th Oct 2019, 1-3pm) (Cover up to Week 6)**
    - PE2(20%) **(9th Nov 2019, 1 to 3pm) (Cover up to Week 9)**
  - Term Test (20%) **(23nd Oct 2019, Lesson time) (Cover up to Week 8)**
- Final exam: (40%)
  - **Wednesday, 27nd Nov 2019, 17:00-19:00**
- All tests and exams are open book(no digital devices)

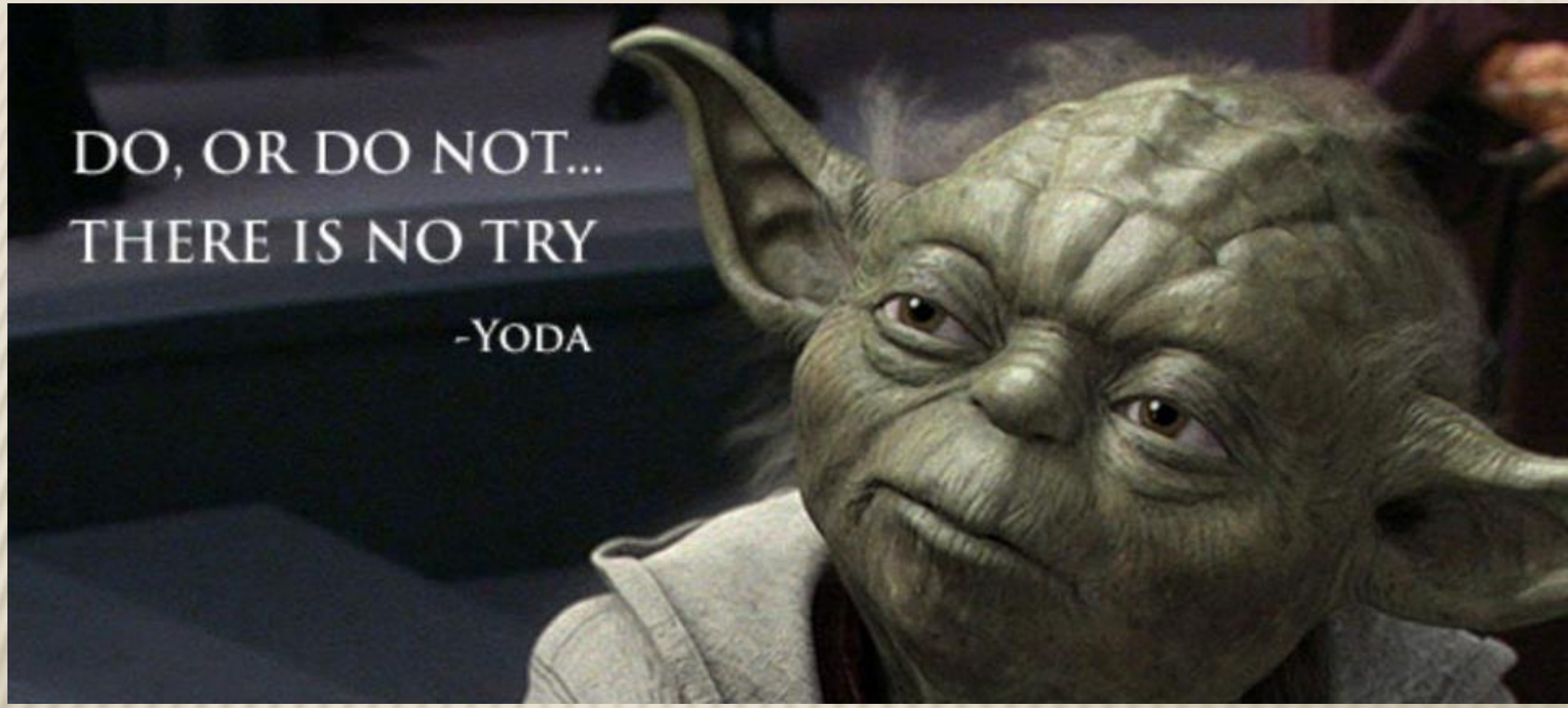
DISCUSSION GROUP

https://t.me/joinchat/H55maBM12a2HFdIU-4kHHg

- Use your real name
- Please restrict discussion to CS1010 topics i.e.
  - Politics
  - BGR
  - Buy and Sell etc. etc.

VIM/UNIX WORKSHOP

- Date: Week 2
  Follow the instruction link to install the ssh client on your laptop
  http://www.comp.nus.edu.sg/~cs1010/2_resources/SSHSecureShellClient-3.2.9.exe
- Create a new unix account at
  - https://mysoc.nus.edu.sg/~newacct
- Bring your laptop so that you can connect to the Unix server to go through the session
- Attendance is NOT compulsory



For Undergraduates	Semester 1 & 2
Add new module(s)	By end of Instructional Week 1
Drop module(s) without grade penalty (50% refund of bid points)	By end of Instructional Week 2
Drop module(s) with 'Withdrawn' Grade (no refund of bid points)	1st day of Instructional Week 3 through last day of Recess Week
Drop module(s) with 'Fail' Grade (full refund of bid points)	1st day of Instructional Week 7 onwards